COBOL for OS/390 & VM
COBOL for MVS & VM

Compiler and Run-Time Migration Guide

GC26-4764-05

COBOL for OS/390 & VM
COBOL for MVS & VM

Compiler and Run-Time Migration Guide

GC26-4764-05

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix P, “Notices” on page 290.

Seventh Edition (September 2000)

This edition replaces and makes obsolete the previous edition, GC26-4764-04. The technical changes for this edition are summarized under “Summary of Changes” and are indicated by a vertical bar to the left of a change.

This edition applies to COBOL for MVS & VM Version 1 Release 2 of 5688-197 and COBOL for OS/390 & VM Version 2 Release 2 of 5648-A25 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, HHX/H3
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

About this book

This book provides information to help you to move your run time to IBM Language Environment for MVS & VM (Language Environment) and to upgrade your source programs to COBOL for OS/390 & VM or COBOL for MVS & VM.

Terminology Clarification

In this book, we use the term IBM COBOL as a general reference to:

- COBOL for MVS & VM Version 1 Release 2
- COBOL for OS/390 & VM Version 2 Release 1
- COBOL for OS/390 & VM Version 2 Release 2

Note: Although this Migration Guide is not intended for upgrading to COBOL/370, in many instances the information in this book also applies to or can be affected by COBOL/370 programs. In such cases, we'll indicate when the information is applicable to COBOL/370.

We use the term Language Environment as a general reference to Language Environment Release 5 and later and the Language Environment element of OS/390.

To aid in moving your run time to Language Environment, this book provides information on how to run existing VS COBOL II and OS/VS COBOL load modules under Language Environment, including link-edit requirements for support and recommended run-time options for compatible behavior.

To aid in upgrading your source programs to the COBOL 85 Standard supported by IBM COBOL, 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 IBM COBOL programs.

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

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 this *Compiler and Run-Time Migration Guide*. Without such assistance, this book would have been much more difficult to develop.

Using your documentation

The publications provided with IBM COBOL and Language Environment are designed to help you do COBOL programming under OS/390 or VM.

COBOL for OS/390 & VM

Figure 1. The COBOL for OS/390 & VM publications

Task	Information	Order number
Evaluate the product	*Fact Sheet*	GC26-9048
Understand warranty information	*Licensed Program Specifications*	GC26-9044
Install the compiler under OS/390	*Installation and Customization under OS/390*	GC26-9045
Install the compiler under VM	*Program Directory for VM*	
Understand product changes—upgrade source to COBOL for OS/390 & VM and run time to Language Environment	*Compiler and Run-Time Migration Guide*	GC26-4764
Prepare and test your programs and get details on compiler options	*Programming Guide*	SC26-9049
Get details on COBOL syntax and specifications of language elements	*Language Reference*	SC26-9046
Diagnose compiler problems not due to user error, and report them to IBM	*Diagnosis Guide*	GC26-9047

COBOL for MVS & VM

Figure 2. The COBOL for MVS & VM publications

Task	Information	Order number
Evaluate the product	*Fact Sheet*	GC26-8664
Understand warranty information	*Licensed Program Specifications*	GC26-4761
Install the compiler under MVS	*Installation and Customization under MVS*	GC26-4766
Install the compiler under VM	*Program Directory for VM*	
Understand product changes—upgrade source to COBOL for MVS & VM and run time to Language Environment	*Compiler and Run-Time Migration Guide*	GC26-4764
Prepare and test your programs and get details on compiler options	*Programming Guide*	SC26-4767
Get details on COBOL syntax and specifications of language elements	*Language Reference*	SC26-4769
Diagnose compiler problems not due to user error, and report them to IBM	*Diagnosis Guide*	SC26-3138

Language Environment Element of OS/390 Version 2 Release 4 or later

Figure 3. The Language Environment element of OS/390 publications

Task	Information	Order number
Evaluate the product	*Concepts Guide*	GC28-1945
Install Language Environment	Language Environment Program Directory	
Customize Language Environment	*OS/390 Customization*	SC28-1941
Understand Language Environment program models and concepts	*Programming Guide*	SC28-1939
Find syntax for Language Environment run-time options and callable services	*Programming Reference*	SC28-1940
Debug applications that run with Language Environment, get details on run-time messages diagnose problems with Language Environment	*Debugging Guide and Run-Time Messages*	SC28-1942
Migrate applications to Language Environment	*Run-Time Migration Guide*	SC28-1944
Develop interlanguage communication (ILC) applications	*Writing Interlanguage Applications*	SC28-1943

Language Environment Release 6 (for VM)

Figure 4. The Language Environment publications for VM

Task	Information	Order number
Evaluate the product	*Concepts Guide*	GC28-1945-02
Install Language Environment	Language Environment Program Directory	
Understand Language Environment program models and concepts	*Programming Guide*	SC28-1939-02
Find syntax for Language Environment run-time options and callable services	*Programming Reference*	SC28-1940
Debug applications that run with Language Environment, get details on run-time messages diagnose problems with Language Environment	*Debugging Guide and Run-Time Messages*	SC28-1942-02
Migrate applications to Language Environment	*Run-Time Migration Guide*	SC28-1944-02
Develop interlanguage communication (ILC) applications	*Writing Interlanguage Applications*	SC28-1943-02

Summary of changes

September 2000—GC26-4764-05

Compiler

- Added newly discovered undocumented extensions and improved many existing entries in Chapter 10, “Modifying OS/VS COBOL source programs” on page 106
- Added new reserved words
- Added information on migrating to the V2R2 compiler

Run time

- Added discussion of the new default for run-time option ABTERMENC (ABEND for Language Environment for OS/390 V2R9 and later) and the new suboptions for TERMTHDACT available in Language Environment for OS/390 V2R7 and later
- Added information about Language Environment region-wide run-time options
- Updated the virtual storage requirements
- Updated the CICS considerations:
  - Performance
  - SORT interface change
  - DISPLAY statement
- Updated information on upgrading Language Environment release levels

Miscellaneous maintenance and editorial changes have been made.

July 1997—GC26-4764-04

Compiler

- Added a high-level overview of the migration process, Year 2000, and service support
- Added the affect of COBOL for OS/390 & VM DLL support for COBOL for MVS & VM object-oriented programs
- Expanded the information on upgrading compiler release levels (from COBOL/370 or COBOL for MVS & VM to COBOL for OS/390 & VM)
- Expanded information on IBM COBOL programs that have static CALLs to VS COBOL II programs
- Added miscellaneous source language differences. For details, see Chapter 10, “Modifying OS/VS COBOL source programs” on page 106
  - ACCEPT statement without the keyword FROM
  - Dynamic CALLs when running on CICS

  - Various MOVE statement differences
  - Periods in Area A
  - RECORD KEY phrase and ALTERNATE RECORD KEY phrase

Run time

- Expanded on virtual storage requirements under Language Environment
- Provided different options for obtaining a system dump or CICS transaction dump
- Added information on the sample user condition handler CEEWUCHA, which provides compatible VS COBOL II behavior under Language Environment. See "Using CEEWUCHA" on page 84 for details.
- Updated ILC information, specifically with COBOL DISPLAY information and the affect of STOP RUN
- Added additional CICS information for WORKING-STORAGE limits and VS COBOL II NORENT programs
- Updated assembler appendix (see Appendix D, "Applications with COBOL and assembler" on page 209), including:
  - Save area requirements when an assembler program is the caller
  - Affect of an MVS ATTACH on the parameter list when an MVS ATTACH is used to invoke a COBOL program
  - Effect on WORKING-STORAGE when an assembler program loads and calls COBOL
  - When assembler cannot SVC delete COBOL programs
  - Storage in subpools
  - AMODE requirements for invoking COBOL programs

Miscellaneous maintenance and editorial changes have been made.

Overview

Chapter 1. Do I need to recompile?

Ideally, programs should be compiled with a supported compiler (recommended is COBOL for OS/390 & VM or COBOL for MVS & VM) and run with a supported run-time library (Language Environment). You can reach this ideal state gradually, in a fully supported manner by starting with a:

1. Run-time migration followed **optionally** at some future date by a
2. Compiler migration

For most programs, you **do not** need to recompile to ensure they continue working properly or have service support. For details on programs that must be upgraded, see:

- "Determining which programs require upgrade" on page 60 for OS/VS COBOL programs running with the OS/VS COBOL run time
- "Determining which programs require upgrade" on page 72 for OS/VS COBOL and VS COBOL II programs running with the VS COBOL II run time
- "Determining which programs require upgrade" on page 159 for COBOL for MVS & VM programs

The remainder of this chapter explains when and why you might want to migrate your applications (run time or source). It includes the following topics:

- Migration basics
- Service support for OS/VS COBOL and VS COBOL II programs

Migration basics

The migration process involves run-time migration (moving your applications to a new run time) and source migration (upgrading your source programs). As part of the migration process, you'll also need to do inventory assessment and testing. As stated previously, you are not required to migrate your run time and source concurrently.

For more details on the migration process, see "General conversion tasks" on page 13.

Run-time migration

Every COBOL program requires run-time library routines to execute. They may be statically linked to the load modules (compiled with the NORES compiler option) or dynamically accessed at execution time (compiled with the RES compiler option).

Moving to Language Environment: If you are starting with load modules consisting of programs compiled with the NORES option and link-edited with the OS/VS COBOL run-time library or the VS COBOL II run-time library, then you will need to use REPLACE linkage editor control statements to replace the existing run-time library routines with the Language Environment versions. If you start with object programs (nonlinked), then you just need to link-edit with Language Environment.

If the programs are compiled with the RES option, make the Language Environment library routines available at execution time in place of the OS/VS COBOL or

VS COBOL II library routines by using LNKLST, LPALST, JOBLIB, or STEPLIB on MVS, and GLOBAL statements on VM.

Do not make more than one COBOL run-time library available to your applications at execution time. For example, there should be one and only one COBOL run-time library, such as SCEERUN for Language Environment, in LNKLST. If you have more than one you will either get hard-to-find errors or you will have an unused load library in your concatenation.

For additional information on moving your run time to Language Environment, see:

- Chapter 5, “Running existing applications under Language Environment” on page 46
- Chapter 6, “Moving from the OS/VS COBOL run time” on page 58
- Chapter 7, “Moving from the VS COBOL II run time” on page 70

Source migration

Source migration is not required for most programs and can occur after you've moved your OS/VS COBOL or VS COBOL II programs to run with Language Environment.

Source migration usually consists of upgrading the source language level used (such as from the COBOL 74 Standard supported by OS/VS COBOL to the COBOL 85 Standard supported by IBM COBOL). One very good reason to update source is to accommodate a four-digit year for programs with date-related logic. Source migration is also required in a few instances to enable your applications to run under Language Environment.

Many conversion tools exist to aid in upgrading your source code. For details, see Appendix C, “Conversion tools for source programs” on page 199.

Service support for OS/VS COBOL and VS COBOL II programs

IBM will continue to provide service support for the execution of programs compiled with the OS/VS COBOL and VS COBOL II compilers when these programs use the Language Environment run-time library versions of the COBOL library routines.

For example, the library routines for OS/VS COBOL programs exist in the OS/VS COBOL, the VS COBOL II, and the Language Environment run-time libraries. OS/VS COBOL programs running with the OS/VS COBOL library are not supported by IBM service. If your OS/VS COBOL programs are running using a supported release of the Language Environment run-time library, your programs are supported by IBM service.

Changing OS/VS COBOL programs: Although the OS/VS COBOL compiler is no longer supported, the programs that were generated by it are supported if they are running under Language Environment. Once you have moved your run time to Language Environment, you can run your source code through a source conversion tool, such as the COBOL and CICS Conversion Aid (CCCA) or the conversion feature of VisualAge for COBOL, and then compile using the IBM COBOL compiler.

For more information on CCCA, see Appendix C, “Conversion tools for source programs” on page 199.

Chapter 2. Introducing the new compiler and run time

This chapter provides an overview of the COBOL compiler, (COBOL for OS/390 & VM and COBOL for MVS & VM) and the common run-time (Language Environment) and introduces you to the terminology used throughout this book. This chapter includes information on the following:

- Product relationships—compiler, run time, debug
- Comparison of IBM COBOL compilers
- Language Environment's run-time support for other programs
- Advantages of the new compiler and run time
- Obstacles to upgrading to the new compiler and run time
- Major changes with the new compiler and run time
- General conversion tasks

Terminology Clarification

In this book, we use the term IBM COBOL as a general reference to:

- COBOL for MVS & VM Version 1 Release 2
- COBOL for OS/390 & VM Version 2 Release 1
- COBOL for OS/390 & VM Version 2 Release 2

We use the term Language Environment as a general reference to Language Environment Release 8 and later and the Language Environment element of OS/390.

What's in a name?: IBM has changed the name of the strategic, host COBOL compiler several times. We apologize for any confusion this may have caused, and here is a brief explanation of why the name has changed.

When we first shipped the follow-on product to VS COBOL II, we followed a new IBM naming convention for adding platform names to the product names, and chose COBOL/370. At about the same time, 370 was changing to 390. Two years later, we were required to add the operating system name to the product names to be even more specific. Thus, COBOL/370 became COBOL for MVS & VM. Finally, MVS became OS/390, so now we have COBOL for OS/390 & VM.

The table below shows the COBOL compiler names, version, release, and product numbers:

Compiler	Release level	Product number
COBOL/370	Version 1 Release 1	5688-197
COBOL for MVS & VM	Version 1 Release 2	5688-197
COBOL for OS/390 & VM	Version 2 Release 1	5648-A25
COBOL for OS/390 & VM	Version 2 Release 2	5648-A25

Product relationships—compiler, run time, debug

COBOL for OS/390 & VM is IBM's strategic COBOL compiler for System 390 platform. COBOL for OS/390 & VM is a superset of VS COBOL II, with additional features such as intrinsic functions, support for a common run-time environment, (Language Environment), object-oriented programming support, LOCAL-STORAGE Section support, and improved interoperability with C.

Language Environment provides a single language run-time environment for COBOL, PL/I, C, and FORTRAN. In addition to support for existing applications, Language Environment also provides common condition handling, improved interlanguage communication (ILC), reusable libraries, and more efficient application development. Application development is simplified by the use of common conventions, common run-time facilities, and a set of shared callable services. Language Environment is required to run IBM COBOL programs.

Debugging capabilities are provided by the Debug Tool. The Debug Tool provides significantly improved debugging function over previous COBOL debugging tools, and can be used to debug IBM COBOL programs, COBOL/370 programs, VS COBOL II programs running under Language Environment, and other Language Environment-conforming language programs.

The Debug Tool is included with the full-function version of the compiler.

Figure 5 shows the relationship between IBM's previous COBOL products and IBM COBOL, Language Environment, and the Debug Tool.

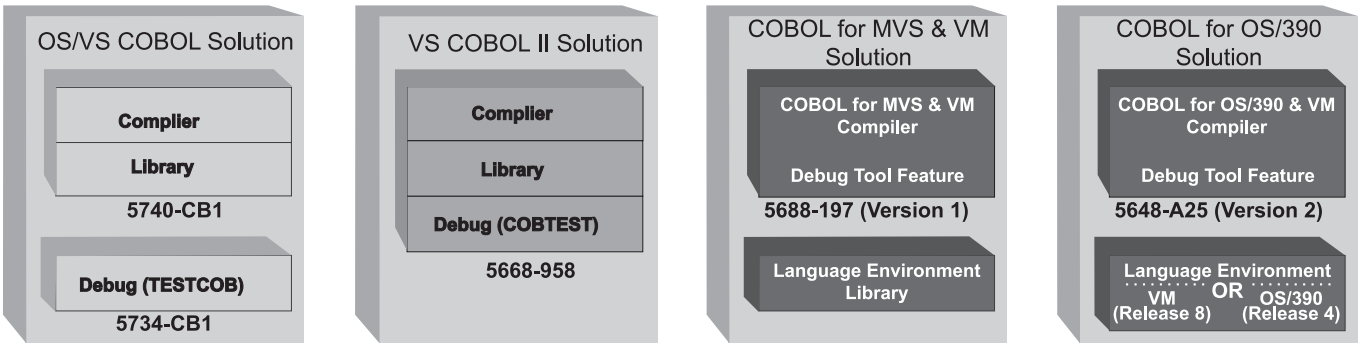


Figure 5. Product relationships—compiler, run time, and debug

Comparison of IBM COBOL compilers

Figure 6 gives an overview of the functions available with the latest releases of OS/VS COBOL and VS COBOL II, and shows the new function available with the IBM COBOL compilers.

Figure 6. Comparison of IBM COBOLs

OS/VS COBOL	VS COBOL II	COBOL for MVS & VM	COBOL for OS/390 & VM
COBOL 74 Standard 74 STD FIPS flagging Dynamic loading Batch debugging Interactive debugging (line mode)	COBOL 74 compatibility 85 STD FIPS flagging Dynamic loading Batch debugging Interactive debugging (line mode)	COBOL 74 compatibility 85 STD FIPS flagging Dynamic loading Batch debugging Interactive debugging (line mode)	COBOL 74 compatibility 85 STD FIPS flagging Dynamic loading Batch debugging Interactive debugging (line mode)
	COBOL 85 Standard No intrinsic funtions Structured programming DBCS National language Improved CICS interface 31-bit addressing Reentrancy, Fast Sort Optimizer, SAA flagging Interactive debugging (full-screen mode)	COBOL 85 Standard Structured programming DBCS National language Improved CICS interface 31-bit addressing Reentrancy, Fast Sort Optimizer, SAA flagging Interactive debugging (full-screen mode)	COBOL 85 Standard Structured programming DBCS National language Improved CICS interface 31-bit addressing Reentrancy, Fast Sort Optimizer, SAA flagging Interactive debugging (full-screen mode)
		Extensions for: Object-oriented COBOL C interoperability Intrinsic Functions Amendment to '85 Std Support for: Language Environment Debug Tool	Extensions for: Object-oriented COBOL C interoperability Intrinsic Functions Amendment to '85 Std Support for: Language Environment Debug Tool
			Support for: DLLs 31 digits DB2 coprocessor OS/390 UNIX Enhanced support for: Debug Tool

The version of the IBM COBOL compiler you can use depends on which host system you are using. Figure 7 lists the IBM COBOL compiler available for each operating system.

Figure 7 (Page 1 of 2). IBM COBOL release levels

IBM COBOL compiler	Language Environment release	
COBOL for MVS & VM	MVS	Language Environment Release 5 or later
	VM	Language Environment Release 5 or later
	OS/390	Language Environment element of OS/390 Release 1 or later
COBOL for OS/390 & VM Release 1	OS/390	Language Environment element of OS/390 Release 3 or later
	VM	Language Environment Release 6 or later

Figure 7 (Page 2 of 2). IBM COBOL release levels

IBM COBOL compiler	Language Environment release	
COBOL for OS/390 & VM Release 2	OS/390	Language Environment Element of OS/390 Version 2 Release 4 or later
	VM	Language Environment Release 8 or later

Note: For a complete list of host versions and releases, see the *License Program Specifications* for Language Environment and the compiler you are using.

Language Environment's run-time support for other programs

Figure 8 shows the programs that are able to run under Language Environment, which provides support for programs that currently run with the OS/VS COBOL and VS COBOL II libraries. Language Environment also provides support for other high-level languages.

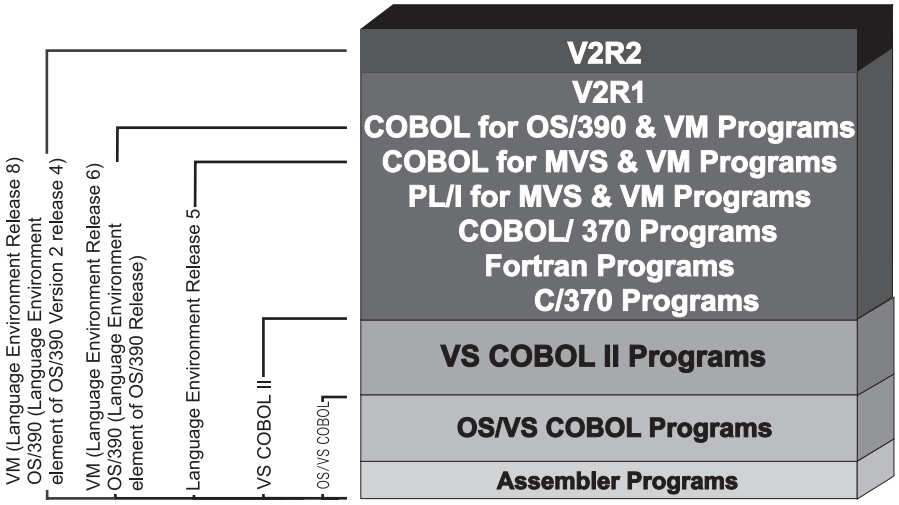


Figure 8. Language Environment's run-time support for other programs

The release of Language Environment you use depends on which host system you are using. Figure 9 lists the version of Language Environment required for each operating system.

Figure 9. Language Environment release levels

Host system[1]	Language Environment release
MVS Version 5	Language Environment Release 5
VM Version 2	Language Environment Release 5 or later
OS/390 Release 1 or later	Language Environment element of OS/390

Note:

1 For a complete list of host versions and releases, see the *License Program Specifications* for Language Environment.

Advantages of the new compiler and run time

The IBM COBOL compiler and Language Environment run time provide additional functions over OS/VS COBOL and VS COBOL II. Figure 10 lists the advantages of the new compiler and run time and indicates whether they apply to VS COBOL II, OS/VS COBOL, or both.

Figure 10 (Page 1 of 3). Advantages of IBM COBOL and Language Environment

		Advantage over	
Advantage	**Notes**	**OS/VS COBOL**	**VS II COBOL**
COBOL language improvements	Ability to perform math and financial functions in COBOL, using Intrinsic Functions. You can replace current routines written in FORTRAN or C with native COBOL code, thus simplifying your application logic.	√	√
Above-the-line support	Virtual Storage Constraint Relief (VSCR) allows your programs to reside, compile, and access programs below or above the 16M line.	√	
	QSAM buffers can be above the 16M line for optimal support of DFSMS and data striping.	√	√
	COBOL EXTERNAL data can now be above the line.	√	√
31-digit support	COBOL for OS/390 & VM Release 2 added support for numbers up to 31 digits when the ARITH(EXTEND) option is used.	√	√
OS/390 UNIX system services support	The cob2 command can be used to compile and link COBOL programs in the OS/390 UNIX shell. COBOL programs can call most of the C language functions defined in the POSIX standard.	√	√
Object-oriented extensions	IBM COBOL provides extensions which enable you to create object-oriented COBOL programs.	√	√
Strategic IBM compiler	Any future performance improvements or language enhancements, for the 370/390 platform, such as Client/Server COBOL, will only be available with COBOL for OS/390 & VM and Language Environment.	√	√
Error recovery options	Programmers now have 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 IBM COBOL programs with Language Environment callable services to register the user-written condition handlers. Language Environment handles all condition management.	√	√
Cost savings	If your shop uses multiple languages, you could see a cost savings by replacing multiple language run times with the single Language Environment run time. Talk with your IBM representative to evaluate the potential cost savings based on the number of current licenses and languages used by your shop.	√	√
High-precision math routines	Using Language Environment callable services, your programs can return the most accurate results.	√	√
Support for multiple MVS tasks	RES applications can now execute independently under multiple MVS tasks. (For example, running two IBM COBOL programs at the same time from ISPF split screens.)	√	√
Performance	Faster arithmetic computations	√	
	Faster dynamic and static CALLs		√
	Improved performance of variable-length MOVEs		√
	Faster CICS performance if using the Language Environment CBLPSHPOP run-time option to prevent PUSH HANDLE and POP HANDLE for CALLs.		√
	Improved performance for programs compiled with TRUNC(BIN). COBOL for OS/390 & VM Release 2 added support to generate more efficient code when the TRUNC(BIN) compiler option is used.		√

Figure 10 (Page 2 of 3). Advantages of IBM COBOL and Language Environment

Advantage	Notes	Advantage over OS/VS COBOL	Advantage over VS II COBOL
Improved ILC	With the common Language Environment library, ILC is improved between COBOL and other Language Environment-conforming languages. For example, interlanguage CALLs between COBOL and other languages are faster under Language Environment, because there is significantly less overhead for each CALL. Additionally, on CICS, you can use the CALL statement to call PL/I or C programs in place of EXEC CICS LINK.	√	√
Character manipulation	Improved bit and character manipulation using hex literals. Improved flexibility with character manipulation using reference modification	√	
Top-down modular program development	Support for top-down modular program development through nesting of programs and improved CALL and COPY functions	√	
Structured Programming Support	Support for structured programming coding through: • Inline PERFORM statements • The CONTINUE place-holder statement • The EVALUATE statement • Explicit scope terminators (for example: END-IF, END-PERFORM, END-READ)	√	
COBOL 85 Standard conformance	Support for the COBOL 85 Standard	√	
	Support for Amendment 1 (Intrinsic Functions Module) of the COBOL 85 Standard	√	√
Subsystem support	Improved support for IMS, ISPF, DFSORT, DB2	√	
Added DB2 function	IBM COBOL includes support for DB2 stored procedures.	√	√
	Support for the DB coprocessor (available in DB2 Version 7)	√	√
Improved CICS interface	IBM COBOL includes CALL statement support (for faster CICS performance than when using EXEC CICS LINK) and eliminates the need for BLL cells. See “Base addressability considerations” on page 161.	√	
	Increased WORKING-STORAGE space for DATA(24) and DATA(31) programs. For DATA(31), the limit is 128M. For DATA(24), the limit is the available space below the 16M line.	√	√
Support for reentrancy	All OS/VS COBOL programs are nonreentrant. Only reentrant programs can be loaded into shared storage (LPA or Shared Segments).	√	
Support for Debug Tool	The Debug Tool provides the following benefits: • Interactive debugging of CICS and non-CICS applications • Interactive debugging of batch applications • Full-screen debugging for CICS and non-CICS applications • Debugging of mixed languages in the same debug session • Ability to debug programs run on the host • Working in conjunction with VisualAge for COBOL, the ability to debug host programs from the workstation using a graphical user interface	√	√

Figure 10 (Page 3 of 3). Advantages of IBM COBOL and Language Environment

Advantage	Notes	Advantage over OS/VS COBOL	Advantage over VS II COBOL
Run-time options	ABTERMENC and TERMTHDACT—allow you to control error behavior.	√	√
	CBLQDA—allows you to control dynamic allocation of QSAM files.		√
	LANGUAGE—allows you to change language of error message.	√	
	RPTSTG—allows you to obtain storage usage reports.	√	
	Storage options—allow you to control where storage is obtained and the amount of storage used.	√	√
Compiler options	The following compiler options are available to COBOL for OS/390 & VM programs only: • DLL - enables the compiler to generate an object module that is enabled for Dynamic Link Library (DLL) support. • EXPORTALL - instructs the compiler to automatically export certain symbols when the object deck is link-edited to form a DLL.	√	√
	The following compiler options are available to COBOL for OS/390 & VM and COBOL for MVS & VM programs: • CURRENCY—Allows you to define a default currency symbol for COBOL programs. • OPTIMIZE(FULL)—OPTIMIZE with the new suboption of FULL optimizes object programs and provides improved run-time performance over both the OS/VS COBOL and VS COBOL II OPTIMIZE options. The compiler discards unused data items and does not generate code for any VALUE clauses for the discarded data items. • PGMNAME(COMPAT,LONGUPPER,LONGMIXED) controls the handling of program names in relation to length and case. • RMODE(AUTO|24|ANY)—allows NORENT programs to reside above the 16M line.	√	√
	IBM COBOL provides, as does VS COBOL II, 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 • Customized reserved word lists • Processing COPY or BASIS statements • Text of error messages • Language of error messages	√	

Obstacles to upgrading to the new compiler and run time

Prerequisite products and function not yet available in Language Environment might delay your upgrade to IBM COBOL and Language Environment. Figure 11 lists the obstacles that might prevent you from upgrading at this time.

Figure 11 (Page 1 of 2). Obstacles to upgrading to IBM COBOL and moving to Language Environment

Enterprise System Architecture (ESA)	VM/ESA or MVS/ESA (or OS/390) are required to run applications under Language Environment. For a complete list of the version and release levels supported, see the *Language Environment Licensed Program Specifications*.

Figure 11 (Page 2 of 2). Obstacles to upgrading to IBM COBOL and moving to Language Environment

CICS Version 4 Release 1	Language Environment requires CICS Version 4 Release 1 or later. CICS macro-level application code must be converted to command-level code to run under CICS Version 4 Release 1 or later.
Other high-level Languages	Currently, Language Environment supports COBOL, C/C++, PL/I, and FORTRAN. ILC with other high-level languages might be limited or nonexistent with Language Environment. (Note, this does not apply to assembler language programs.)
Vendor product support	Any vendor products used by your shop should be Language Environment-conforming. Language Environment-conforming means: **For a TOOL** A program that runs correctly in a Language Environment environment with the TRAP(ON) run-time option in effect. If you have a tool that issues an ESPIE or ESTAE, it must coordinate with Language Environment. **For a PACKAGE** A program that either runs in a Language Environment environment in compatibility mode, or is written in a language that can be compiled by a Language Environment-conforming compiler. Contact your vendors to verify that their products are Language Environment-conforming. To obtain information on which vendor tools are Language Environment-conforming, or are in plan to be Language Environment-conforming, see "Vendor products" on page 208.

Suggestions for incremental conversions

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

- Evaluate the effort to move to Language Environment.
- Develop a conversion plan and long-range schedule.
- Convert your macro-level CICS programs to command-level programs. (For guidance information, see "CICS Application Migration Aid" on page 206.)
- Code applications based on IBM COBOL and Language Environment requirements to ease a future conversion. For example, specify the RES compiler option instead of the NORES compiler option; all IBM COBOL programs are RES. A "RES-conversion" is easier than a "NORES-conversion." Link-editing NORES applications with Language Environment can result in different behavior, depending on how the programs in the application are coded. For details, see Chapter 8, "Link-editing applications with Language Environment" on page 99.
- Determine which applications contain ILC between C and VS COBOL II. Verify that the COBOL and C used are supported by Language Environment and make the appropriate changes as described in the *C Migration Guide*. For details, see "COBOL and C/370" on page 91.
- Determine which applications contain ILC between PL/I and VS COBOL II. Verify that the COBOL and PL/I used are supported by Language Environment, and link-edit the PL/I programs with the PL/I migration tool. For details, see "COBOL and PL/I" on page 91.

The PL/I migration tool allows you to link-edit your PL/I programs gradually, while still running on the PL/I run time. If you link-edit with the migration tool, you do not have to link-edit before running under Language Environment.

- Convert all COBOL source code to COBOL 85 Standard.
- Upgrade to VS COBOL II Release 4 and change your VS COBOL II reserved word table to restrict the use of reserved words added in VS COBOL II and IBM COBOL (as listed in "Reserved words" on page 268). Then, you will not need to make any source changes to compile with IBM COBOL.

  Moving from the VS COBOL II run time to Language Environment is much easier than moving from the OS/VS COBOL run time to Language Environment. However, some VS COBOL II features are not supported in Language Environment. When upgrading to VS COBOL II, avoid the following:

  – Compiling programs with the NORES compiler option
  – Tuning programs using the IGZTUNE macro
  – Creating a BLDL user exit (IGZEXIT)
  – Using IGZEOPT to specify application-specific run-time options on CICS

  Note: When you move to VS COBOL II Release 4, make sure that you have APAR PN74000 and APAR PN65736 applied.

  Although this path requires two conversions, if you follow the coding techniques mentioned above, moving from VS COBOL II Release 4 to Language Environment will require minimal, if any, changes.

Major changes with the new compiler and run time

With Language Environment, you will find that existing COBOL applications are affected in two areas: application termination behavior and the NORES environment. A brief description of the differences and actions required to ensure compatibility follow.

Application termination behavior: When running with the VS COBOL II run time or the OS/VS COBOL run time, severe errors (such as program checks and abends) resulted in an abend to end the application.

With Language Environment, when a severe error occurs, the termination behavior is controlled by the ABTERMENC run-time option. For all releases of Language Environment prior to Version 2 Release 9, the IBM-supplied setting for non-CICS is ABTERMENC(RETCODE). With this setting, when a severe error occurs, applications will end normally with a return code (instead of abending).

Note: On non-CICS, to ensure that your applications end with an abend when there is a severe error, you must change the default setting of ABTERMENC to ABTERMENC(ABEND) when you are using a release of Language Environment prior to Version 2 Release 9.

NORES environment not available: IBM COBOL does not provide the NORES compiler option. All IBM COBOL programs are RES. Existing NORES applications are not affected; they continue to run and provide the same results.

If you link-edit existing NORES programs with Language Environment, or add an IBM COBOL program to a NORES application, you have additional considerations. For details, see:

- Chapter 8, “Link-editing applications with Language Environment” on page 99
- Chapter 16, “Adding IBM COBOL programs” on page 170

General conversion tasks

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

- Plan your strategy
- Move your run time to Language Environment
- Upgrade your source to IBM COBOL
- Add new COBOL programs to existing applications

Plan your strategy

Before moving your run time to Language Environment and upgrading your source programs to IBM COBOL, develop a conversion strategy. A thorough strategy will help ensure a smooth transition to the new compiler and run time.

Your conversion strategy might be to move to Language Environment, and then gradually recompile your existing applications with IBM COBOL as needed. This book provides separate strategies for moving to the new run time and for upgrading source. For details, see:

- Chapter 3, “Planning the move to Language Environment” on page 18
- Chapter 4, “Planning to upgrade source programs” on page 31

Move to the Language Environment run time

You can run existing load modules under Language Environment and receive the same results as with your current run-time library. For important compatibility information, see Chapter 5, “Running existing applications under Language Environment” on page 46.

For information on moving applications that are running under the OS/VS COBOL run time, see Chapter 6, “Moving from the OS/VS COBOL run time” on page 58.

For information on moving applications that are running under the VS COBOL II run time, see Chapter 7, “Moving from the VS COBOL II run time” on page 70. (Since you can run OS/VS COBOL programs under the VS COBOL II run time, this chapter duplicates the information from Chapter 5 that is applicable to OS/VS COBOL programs running under the VS COBOL II run time.)

In some cases, you will need to link-edit existing applications with Language Environment or upgrade programs to IBM COBOL. To determine which programs require link-edit with Language Environment, see:

- “Determining which programs require link-edit” on page 59 for programs running under the OS/VS COBOL run time
- “Determining which programs require link-edit” on page 71 for programs running under the VS COBOL II run time

Link-editing programs with Language Environment can result in different behavior, see Chapter 8, “Link-editing applications with Language Environment” on page 99 for details.

To determine which programs must be upgraded to IBM COBOL , see:

- "Determining which programs require upgrade" on page 60 for programs running under the OS/VS COBOL run time
- "Determining which programs require upgrade" on page 72 for programs running under the VS COBOL II run time

Upgrade your source to IBM COBOL

The effort required to upgrade your source programs is dependent on the compiler used and the language level used. In addition to any source changes required for upgrading language levels, you should also examine your applications to determine if they have date-related logic that will be impacted by the Year 2000. You can then make the changes necessary for the Year 2000 at the same time that you upgrade your COBOL source.

OS/VS COBOL

OS/VS COBOL programs compiled with either LANGLVL(1) or LANGLVL(2) can contain either COBOL 68 Standard or COBOL 74 Standard elements. Conversion is required in order for these programs to compile with IBM COBOL. You should use conversion tools to aid in this conversion. For details, see "Using conversion tools to convert programs to COBOL 85 Standard" on page 110.

VS COBOL II

From a conversion standpoint, the only language difference between VS COBOL II Release 4 and IBM COBOL is the addition of new reserved words. However, if you select the NOOO alternate reserved word table, the compiler will ignore the new words reserved for object-oriented COBOL. A complete list of reserved words, including those reserved for object-oriented COBOL is included in Appendix B, "COBOL reserved word comparison" on page 190. The new reserved words are in **boldface**.

If upgrading from VS COBOL II Release 3, there are also three minor language differences due to ANSI interpretation changes. Aside from these small differences, you can compile with IBM COBOL without change and receive the same results. For details, see Chapter 12, "Modifying VS COBOL II source programs" on page 152.

VS COBOL II Release 2 programs are coded to the COBOL 74 Standard as are VS COBOL II programs compiled with the CMPR2 compiler option. To compile these programs with NOCMPR2 under IBM COBOL, source conversion is required. Conversion tools can help you upgrade your source programs to the COBOL 85 Standard. Details of language differences between CMPR2 and NOCMPR2 are included in Appendix L, "Differences between CMPR2 and NOCMPR2" on page 247.

For details on the conversion tools available to upgrade source programs, see Appendix C, "Conversion tools for source programs" on page 199.

Add new COBOL programs to existing applications

You can create new IBM COBOL programs (or recompile existing programs with IBM COBOL) and run them with existing applications under Language Environment.

When adding IBM COBOL programs to existing applications, you need to be aware of the effects of link-editing with Language Environment, the restrictions of running programs above or below the 16M line, the effect of compiler option changes, reserved word changes, and other behavior differences with IBM COBOL. For details, see Chapter 16, “Adding IBM COBOL programs” on page 170.

Conversion strategies

Chapter 3. Planning the move to Language Environment

This chapter describes a general strategy for moving your run-time environment to Language Environment. The following tasks are necessary, and should be performed in roughly the following order:

1. Prepare to move your run time to Language Environment
2. Take an inventory of your applications
3. Decide how to phase in Language Environment
4. Set up a regression testing procedure
5. Cut over to production use

Important!

On VM, COBOL for OS/390 & VM Release 1 programs can run only under Language Environment Release 6 or later. On OS/390, COBOL for OS/390 & VM programs can run only with the Language Environment element of OS/390 Release 3 or later.

On VM, COBOL for OS/390 & VM Release 2 programs can run only under Language Environment Release 8. On OS/390, COBOL for OS/390 & VM Release 2 programs can run only with the Language Environment element of OS/390 Version 2 Release 4 or later.

COBOL for MVS & VM programs run under Language Environment Release 5 and later.

Prepare to move your run time to Language Environment

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

- Install Language Environment
- Educate your programmers about Language Environment
- Assess storage requirements

Install Language Environment

To install Language Environment, see:

- On VM: *Language Environment Program Directory*
- On MVS: *Language Environment Installation and Customization*
- On OS/390: *OS/390 Program Directory*

Important: To ensure the Language Environment 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 Language Environment run-time options” on page 46.

Assess storage requirements

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

DASD storage requirements

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

To determine the amount of DASD storage required by Language Environment, see:

- On VM: *Language Environment Program Directory*
- On MVS: *Language Environment Installation and Customization*
- On OS/390: *OS/390 Program Directory*

Virtual storage requirements

Virtual storage requirements for running COBOL programs with Language Environment will increase over both the OS/VS COBOL run time and the VS COBOL II run time. For both CICS and non-CICS applications, the amount of increase depends on many factors, such as:

- The values used for the Language Environment run-time storage options: STACK, LIBSTACK, HEAP, ANYHEAP, BELOWHEAP.

  Note: You can use the information generated by the Language Environment RPTSTG run-time option to help tune your storage options. For details, see the *Language Environment Programming Reference*.

- The value used for the Language Environment run-time option ALL31.
- Which run-time routines are in the LPA or the extended LPA
- Additionally, when moving from the VS COBOL II run-time:
  - VS COBOL II run-time options specification
  - If the COBPACKs have been modified to run above the 16M line.

Important

The virtual storage data presented in the following sections was obtained by running sample programs in a particular hardware and software configuration using a selected set of tests and are presented for illustration purposes only. It is recommended that you run your own programs with a configuration applicable to your environment to determine what impact Language Environment will have on your virtual storage.

Virtual storage information for non-CICS: Figure 12 shows how much virtual storage is used when a simple VS COBOL II program compiled RES was run on MVS and OS/390. In each case, the IBM-supplied defaults are used and the run-time library routines were acquired from STEPLIB:

Figure 12. Virtual storage used by a VS COBOL II RES program on non-CICS

Run-time library	Modifications from IBM default	Below 16M virtual storage	Above 16M virtual storage
VS COBOL II Release 4	None	276K	4K
VS COBOL II Release 4	COBPACKs modified to run above the 16M line	32K	192K
Language Environment Release 5	None	268K	1032K
Language Environment element of OS/390 Version 2 Release 7	None	268K	1176K

When comparing the amount of virtual storage used by the Language Environment run time and your current run time, note that run-time routines accessed from the LPA or extended LPA are not included in the virtual storage used by the job.

Virtual storage information for CICS: Language Environment uses more CICS dynamic storage area than the VS COBOL II run time. The amount of CICS dynamic storage area used and whether the storage is allocated from CICS UDSA (dynamic storage area below the 16M line) and from CICS EUSDA (dynamic storage area above the 16M line) depends on many factors.

The most important factors include:

- The value used for the Language Environment run-time option ALL31. **Using ALL31(OFF) as an installation default on CICS can cause Language Environment to use a significant amount of below-the-line CICS user dynamic storage area (UDSA)**.
- The values used for the Language Environment run-time storage options: STACK, LIBSTACK, HEAP, ANYHEAP, BELOWHEAP.
- The nesting level of CICS LINKs. Each CICS LINK starts a new run unit. Hence, the deeper the nesting level, the more storage Language Environment will use.
- The COBOL compiler used. The CICS dynamic storage area used to run programs compiled with the OS/VS COBOL compiler will not increase because when an OS/VS COBOL program is run on CICS, the environment that is established for a run unit by the compatibility run-time library routines supports OS/VS COBOL.

Language Environment uses more CICS dynamic storage area than the VS COBOL II run time for the following reasons:

1. Storage management is done per run unit with Language Environment. Thus, STACK, LIBSTACK, and the different heaps are allocated per run unit. With VS COBOL II STACK (SRA) and heap were managed at the transaction level.
2. Additional control blocks are allocated by the common component of Language Environment.

The following shows storage usage data for the VS COBOL II run time and Language Environment run time on CICS Version 4. The data was collected by running a simple transaction where one VS COBOL II program does a EXEC CICS LINK to another VS COBOL II program with auxiliary trace turned on. The amount of storage used was determined by looking at the GETMAIN trace entries in the auxiliary trace output.

Figure 13. Storage allocation for CICS applications

Storage allocated	VS COBOL II Run time	Language Environment[1] Run time
Per transaction	2040 bytes Storage is below the line if the transaction is defined with TASKDATALOC(BELOW). Storage is above the line if the transaction is defined with TASKDATALOC(ANY). Fixed size used for control blocks.	14052 bytes Storage is below the line if the transaction is defined with TASKDATALOC(BELOW). Storage is above the line if the transaction is defined with TASKDATALOC(ANY). Note, with CICS Version 4 with APAR PN85951 applied, or CICS Transaction Server, the storage will always be allocated above the line. Fixed size used for control blocks.
Per run unit	740 bytes below the line (400 bytes for GETMAIN and 340 bytes for control blocks) On the first run unit, above and below the line heap storage is allocated. This storage is used by any new run units created during the transaction. Additional storage is allocated as needed. When using the IBM-supplied defaults, 8168 bytes of below-the-line storage and 16352 bytes of above-the-line storage is allocated when the first run unit starts.	
Per run unit with ALL31(ON)		29240 bytes above the line. This includes storage for control blocks, plus storage for STACK, LIBSTACK, HEAP, and ANYHEAP. If any below the line heap is needed, it is allocated on demand.
Per run unit with ALL31(OFF) and STACK(4K,4K,BELOW,KEEP)		12208 bytes above the line. This includes storage for control blocks, plus storage for HEAP and ANYHEAP. 17872 bytes below the line. This includes storage for control blocks, plus storage for LIBSTACK, STACK, and BELOWHEAP.

Note:

1 This scenario was run on Language Environment Version 2 Release 7 using the IBM-supplied values for run-time options: STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP.

Educate your programmers about Language Environment

Before moving your run time to Language Environment, ensure that your application programmers are familiar with the features of Language Environment and the differences between your current run-time environment and the Language Environment run time.

Once your programmers are familiar with Language Environment, they can better prepare for the move to Language Environment. For example, they can assist in

taking an inventory of applications. They can also code according to the suggestions listed in “Suggestions for incremental conversions” on page 11.

For information on IBM COBOL and Language Environment education available through IBM, you can call 1-800-IBM-TEACH. You can also get information directly from Language Environment publications, from user groups (such as SHARE), and from the Web at http://www.ibm.com/s390/le.

Take an inventory of your applications

In planning your move to the Language Environment run time, you need to take a comprehensive inventory of the applications you intend to run on Language Environment. Include in this inventory:

- Vendor tools, packages, and products
- COBOL applications

The Edge Portfolio Analyzer can aid in taking an inventory of your existing load modules, see “The Edge Portfolio Analyzer” on page 208 for more information.

Vendor tools, packages, and products

Before you can begin moving your run time to Language Environment, you need to know if your vendor tools, packages, and products are designed to run under Language Environment. Verify that:

- All packages will run under Language Environment, especially if you do not get the source code for them.
- Source code for packages, if you do get the source, is IBM COBOL-compatible source code (1985 Standard-level COBOL).
- Code generators generate IBM COBOL-compatible source code (1985 Standard-level COBOL).
- Development tools and debuggers that issue their own ESPIE or ESTAE coordinate with Language Environment.

For information on how to obtain a list of vendor products that are enabled for Language Environment, see “Vendor products” on page 208.

COBOL applications

When taking an inventory of your COBOL applications, you need to gather information on the program attributes that affect moving to Language Environment. This includes how and what to test and what will affect performance under Language Environment. For your inventory, determine:

For moving your applications to Language Environment:

- Which programs have been compiled with VS COBOL II and which with OS/VS COBOL
- Which programs have been compiled with RES and which with NORES
- Run-time options used (and how specified), specifically:
  - MIXRES (VS COBOL II)
  - RTEREUS (VS COBOL II)
  - LIBKEEP (VS COBOL II)

  - FLOW (OS/VS COBOL)
- Which COBOL programs CALL or are called by assembler programs
- Which COBOL programs use interlanguage communication (PL/I, C, or FORTRAN)
- Which COBOL programs are used under CICS, IMS, DB2, or other subsystems
- Control statements used
- 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)

Assign complexity ratings

This section lists the program attributes that can require changes when moving to the Language Environment run time. Although, in the following description each action within a range indicates "or," one or more of the actions might actually apply to a given attribute.

Each program attribute is assigned a complexity rating, defined as:

0-2 Requires minimum testing, or
Runs under Language Environment without change and without link-editing with Language Environment

3-6 Requires moderate testing, or
Requires moderate coordination, or
Requires link-edit with Language Environment

7-10 Requires moderate to high degree of testing, or
Requires moderate to high degree of coordination, or
Requires rewrite of module, or
Does not run under Language Environment

When moving from OS/VS COBOL run time

Figure 14 shows estimated complexity ratings for conversions of specific program attributes:

Figure 14 (Page 1 of 2). Complexity ratings for programs running under the OS/VS COBOL run time

Program attribute	Complexity rating
Compiled with NORES link-edited with OS/VS COBOL	0
CICS online	1
Link-editing programs using ILBOSTP0 with the assembler driver	3
IMS online	5
Mixed RES and NORES programs	5

Figure 14 (Page 2 of 2). Complexity ratings for programs running under the OS/VS COBOL run time

Program attribute	Complexity rating
ISPF program	7
Called by assembler routine with a LINK SVC	7
CICS (if using dynamic CALLs from OS/VS COBOL programs)	7
Assembler programs that do not follow normal save area conventions. For details, see "Determining requirements for calling and called assembler programs" on page 209.	8
ILC between OS/VS COBOL programs and PL/I programs	8
ILC between OS/VS COBOL programs and FORTRAN programs	8
Multiple load module application, where the main module has OS/VS COBOL NORES program(s) and no IBM COBOL or VS COBOL II programs are included in the main module.	10
Uses QUEUE run-time option	10
Assembler programs that issue a STAE or SPIE	10
Calls assembler routines that do not have valid 31-bit addresses in Register 13 for save area changes.	10
Assembler programs that are coded based on the internals of the ILBO routines	10

For additional details, see:

- Chapter 6, "Moving from the OS/VS COBOL run time" on page 58
- Appendix D, "Applications with COBOL and assembler" on page 209

When moving from the VS COBOL II run time

Figure 15 shows estimated complexity ratings for conversions of specific program attributes.

Figure 15 (Page 1 of 2). Complexity ratings for programs running under the VS COBOL II run time

Program attribute	Complexity rating
OS/VS COBOL programs compiled NORES running under VS COBOL II	0
VS COBOL II programs compiled NORES running under VS COBOL II	0
Loading IGZERRE from an assembler driver	0
Use of IGZEOPT object module (for non-CICS applications)	1
ILC between VS COBOL II programs and PL/I programs, if you link-edit the programs using the PL/I migration tool.	1
CICS online	2
Relies on behavior of RTEREUS run-time option	2
Use of IGZEOPT object module (for CICS applications)	2
Called by assembler routine with a LOAD and Branch	2
ISPF program	3

Figure 15 (Page 2 of 2). Complexity ratings for programs running under the VS COBOL II run time

Program attribute	Complexity rating
Use of IGZETUN object module	3
Compiled with NORES and specifies MIXRES run-time option	4
IMS online	4
ILC between VS COBOL II programs and C/370 programs	4
ILC between VS COBOL II programs and PL/I programs	4
ILC with FORTRAN programs	4
Link-editing programs using IGZERRE with the assembler driver	4
Use of ILBOSTP0 with the assembler driver	4
Specifies MIXRES run-time option for OS/VS COBOL programs without using IGZBRDGE	6
CMS program invoked with START command	7
Use of BLDL user exit	8
Assembler programs that do not follow normal save area conventions For details, see “Determining requirements for calling and called assembler programs” on page 209.	8
ILC between OS/VS COBOL programs and PL/I programs	8
Assembler program LINKs to a COBOL program when running under a reusable environment	9
Assembler programs that issue a STAE or SPIE	10

For additional details, see:

- Appendix D, “Applications with COBOL and assembler” on page 209
- Chapter 7, “Moving from the VS COBOL II run time” on page 70

Set up conversion/no-conversion categories

After you have determined the amount of effort required to move existing load modules to Language Environment, and taken into account program importance and frequency of execution, you can list your programs in the order you want to move them to Language Environment.

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

- Load modules that require the use of the PL/I, C, or FORTRAN portion of Language Environment.
- OS/VS COBOL programs that use ILC with FORTRAN or PL/I

For these cases, STEPLIB to your existing run-time until you can rewrite the applications.

Decide how to phase Language Environment into production mode

When you are ready to use Language Environment in production mode, you need to:

- Determine how to handle multilanguage conversion
- Determine how applications will have access to the library

Multilanguage conversion

If you have COBOL applications with ILC, move them to the Language Environment run time after you have converted each of the languages involved. For example, move a COBOL-PL/I application to Language Environment after you have moved your COBOL-only and PL/I-only applications to Language Environment.

Note: Do not install two different libraries for a given language in LNKLST/LPALST. For example, if you install Language Environment with the COBOL component in LNKLST/LPALST, do not have the OS/VS COBOL library or the VS COBOL II library installed in LNKLST/LPALST.

Once Language Environment is installed in LNKLST, all of your COBOL applications will run under Language Environment by default. If you have programs that need to access the OS/VS COBOL or VS COBOL II run times, you must use STEPLIB or JOBLIB.

Determine how applications will have access to the library

Two general methods are available for moving Language Environment into production: adding Language Environment to the LNKLST/LPALST or using a STEPLIB approach. These scenarios are intended for programs run on OS/390 or MVS.

LNKLST/LPALST

Once you add Language Environment to the LNKLST/LPALST, Language Environment is available to all of your applications. To ensure all applications are functioning correctly under Language Environment before adding Language Environment to your LNKLST/LPALST, you can temporarily install Language Environment in LNKLST/LPALST or use STEPLIB.

Do not make more than one COBOL run-time library available to your applications at execution time. For example, there should be one and only one COBOL run-time library, such as SCEERUN for Language Environment, in LNKLST. If you have more than one, you will either get hard-to-find errors or you will have an unused load library in your concatenation. When you add Language Environment to LNKLST/LPALST, remove any other COBOL run-time libraries, such as COBLIB and COB2LIB.

Temporary installation in LNKLST/LPALST or use STEPLIB: Suggestions for temporarily installing Language Environment in LNKLST/LPALST include:

- "ZAP" Language Environment into LNKLST/LPALST using a vendor product and verify that your applications run under Language Environment. For information on how to obtain a list of vendor products, see "Vendor products" on page 208.

- IPL over a weekend and install Language Environment in LNKLST/LPALST. Verify over the weekend that your applications run under Language Environment.
- Install Language Environment in LNKLST/LPALST on a test or development machine first.
- Use the SETPROG MVS system command to temporarily modify the LNKLST or LPA, without having to IPL the system. For information on using the SETPROG command, see *OS/390 MVS System Commands*, GC28-1781.

OS/390 Note: Although many elements of OS/390 depend on the Language Environment run-time library, OS/390 does not require Language Environment to be installed in LNKLST. (However, Language Environment must be installed in the same zone as OS/390.) If you choose not to place Language Environment in LNKLST, you must STEPLIB Language Environment in the individual OS/390 PROCs that required Language Environment. For information on which elements require Language Environment, see:

- The OS/390 Program Directory for OS/390 Release 4 or later.
- Information APAR II10425 for OS/390 Release 1, 2, and 3.

STEPLIB

You can choose to phase in Language Environment gradually by using the STEPLIB approach. When you STEPLIB to the Language Environment run time, you phase in one region (CICS or IMS), batch (group of applications), or user (TSO), at a time.

Although using STEPLIB means changing JCL, a gradual conversion can be easier than moving all of your applications at one time. Also note that when using STEPLIB, programs will run slower than when they access the run-time library through LNKLST/LPALST and more virtual storage will be used.

Note: 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.
In addition to revising your JCL to STEPLIB to the Language Environment run time during initial setup, you might also need to specify CEEDUMP DD if the default allocation for CEEDUMP does not meet your shop's needs. (CEEDUMP is the ddname where Language Environment writes its dump output.) For default CEEDUMP destinations, see “Language Environment formatted dumps” on page 85.

Problems with STEPLIB and IMS programs

When using STEPLIB on IMS/DC online to access the Language Environment run time, any Language Environment library routines that you have preloaded will not be loaded into read-only storage. If your application has an error and over-writes nonapplication storage, preloaded run-time routines can become corrupted and eventually cause abends when used. At refresh time, these preloaded routines marked reentrant are not refreshed unless loaded from the LPA or the LNKLST/LPALST. Thus, the abends will recur.

Note: This is a 20-year old problem with MVS (OS/390), IMS, and STEPLIB, and is mentioned here because of the proposed STEPLIB approach for gradually moving to Language Environment.

You can use either of the following methods to prevent this problem:

- Install Language Environment into the LNKLST/LPALST.
- Do not preload any run-time routines. (This will slow performance.)

How to minimize the impact:

- Keep your certification of Language Environment as short as possible. (The sooner it is certified, the sooner you can install in LNKLST/LPALST.)
- Watch for different applications abending in the same region, which would indicate that you need to follow the recovery procedure.

How to recover: If you do notice several different applications abending in the same region, stop the region and restart with these IMS commands:

1. Determine the region number by issuing: `'/DISPLAY ACTIVE'`
2. Stop the region by issuing: `'/STOP REGION region#'`
3. Restart the region by issuing: `'/START REGION region-name'`

STEPLIB example

Here is one example of how to phase in Language Environment using the STEPLIB method: for an organization that has a central development center (all compiling and linking is done in one location) and separate production sites. This is a very conservative approach, but it has been used by many customers who require absolutely no disruption in production applications.

1. Certify Language Environment and IBM COBOL at the central development center.
   - Run tests with captured data on your current run time, save all results.
   - Install Language Environment in a STEPLIB environment. This means that unchanged jobs will run with your current run time, and that some users can use the Language Environment run time by using STEPLIB JCL to access the Language Environment run-time library.

     Note for NORES Applications: This section does not apply to NORES applications that have not been changed. However, if you change your NORES applications (for example, by link-editing them with Language Environment), they might behave differently than before the link-edit.
   - Run tests with captured data on the Language Environment run time, using the STEPLIB environment, and compare the results to your current run time. Run parallel tests throughout the certification cycle to ensure that your applications produce the same results when run with Language Environment as they did with your current run time.
   - Finally, compile your test applications using IBM COBOL. STEPLIB to the Language Environment run-time library, and rerun the certification tests.
2. Install Language Environment on the central development center's system and test.
   - Run parallel tests of the nonconverted versions of your existing applications using STEPLIB to access your current run time.
   - Run all new applications in the Language Environment run-time environment before releasing to production runs.
3. Prepare a backout strategy

- Save the procedures for installing your current run time in case you need to backout the Language Environment run time.

4. Install the Language Environment run time at one production site.
   - Continue to run parallel tests of the nonconverted versions of your existing applications with your current run time in the STEPLIB environment.
   - Run the Language Environment run time for one month at this production site.
5. Install the Language Environment run time at all production sites.
   - Optional: continue to run parallel tests of the nonconverted versions of your existing applications with your current run time in the STEPLIB environment.
   - Run the Language Environment run time for one month at all production sites.
   - After one month, delete all of your current run-time library.
6. Compile all new or changed applications with IBM COBOL.

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

Set up a regression testing procedure

Although most applications will run under Language Environment with the same results as on their existing run time, results could differ depending on coding styles, resource utilization, performance, abend behavior, or more strict adherence to IBM conventions in Language Environment. Two examples of where results could differ include applications that use assembler programs that use an SVC LINK instead of COBOL dynamic CALLs and applications that have native COBOL CALLs to OS/VS COBOL programs under CICS.

Since there are so many possible combinations of coding techniques, the only way to determine if your applications will run under Language Environment and receive the expected results, is to set up a procedure for regression testing. Move your applications to a test environment, and ensure you receive the expected results when running under Language Environment.

Regression testing will help to identify if there are:

- OS/VS COBOL programs using assembler stubs with SVC LINK in place of COBOL dynamic CALLs.
- Unclosed files opened by OS/VS COBOL programs or non-COBOL programs, which cause a C03 abend.
- Under CICS, OS/VS COBOL programs using dynamic CALLs.
- Under CICS, VS COBOL II programs using the CALL statement to CALL OS/VS COBOL programs.
- Storage usage differences between your current run time and the Language Environment run time.

- CPU time differences between your current run time and the Language Environment run time.

During testing, run your existing applications in parallel on both your current run time and under the Language Environment run time to verify that the results are the same. Take performance measurements of your existing applications to compare with Language Environment. You can temporarily add Language Environment to the LNKLST/LPALST (either by using a vendor product to "ZAP" Language Environment into the LNKLST/LPALST or by installing Language Environment in the LNKLST/LPALST over a weekend).

Once the program runs correctly, test it separately and also test it with other programs in a run unit. By testing it against a variety of data, 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 Debug Tool or Language Environment dump 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.

Take performance measurements: Once your applications are running under Language Environment in a test environment, take performance measurements—especially on any time-critical or response-critical applications.

After you compare run-time performance between Language Environment 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 Language Environment run-time options. For additional information, see the performance information available on the COBOL Web site. Go to the Library Section at:

`http://www.ibm.com/software/ad/cobol/`

Cut over to production use

When your testing shows the entire application (or group of applications, if running more than one application in an IMS region, or on TSO) receives the expected results, you can move the entire unit over to production use. However, in case of unexpected errors, be prepared for instant recovery:

- Under OS/390, MVS, and VM/ESA, run the old version as a substitute from the latest productivity checkpoint.
- Under DB2, CICS, and IMS, return to the last commit point and then continue processing from that point using the unmigrated COBOL program. (For DB2, use an SQL ROLLBACK WORK statement.)
- For batch applications, use your shop's backup and restore facilities to recover.

After you move your existing applications to production use under the Language Environment run time, monitor your applications for a short time to ensure that they continue to work properly. Then, you can run with the confidence that you had in your previous run time.

Chapter 4. Planning to upgrade source programs

This chapter describes a general strategy for upgrading your source programs to IBM COBOL. The following tasks are necessary, and should be performed in roughly the following order:

1. Prepare to upgrade your source
2. Take an inventory of your applications
3. Make application program updates

Due to the loss of service support for older COBOL compilers, you should eventually upgrade all of your COBOL source programs. Although this is not an immediate requirement, it is very likely that in the future, the older compilers will not run and no fixes will be available. At that point, you will be forced to do a 'quick' migration, and this might be at a very inconvenient time.

Before you upgrade your source programs, you must move your applications to Language Environment.

Prepare to upgrade your source

In preparing to upgrade your source to IBM COBOL, you need to perform the following tasks, which can be done concurrently:

- Install IBM COBOL
- Assess storage requirements
- Decide which conversion tools to use
- Educate your programmers on new compiler features

Install IBM COBOL

If you haven't already done so, install the compiler.

For MVS or OS/390, see the *Installation and Customization* for your product. For VM, all the information you need to install the compiler under VM is included in the *COBOL Program Directory*.

Assess storage requirements

You can load most of the IBM COBOL compiler above the 16M line. In addition, IBM COBOL object programs execute in 31-bit addressing mode and can reside above the 16M line, which frees storage below the 16M line. You can use the freed storage for programs or data that must reside below the line.

During conversion, you will need DASD storage for your current COBOL compiler(s) as well as for the IBM COBOL compiler. When you have completed conversion, and if you have upgraded all of your OS/VS COBOL and/or VS COBOL II programs to IBM COBOL, you will be able to free the storage reserved for your current COBOL compiler. You can choose to keep your OS/VS COBOL compiler, but only the run-time library will have service support under Language Environment.

The load module produced from the same source code when compiled with IBM COBOL will probably be larger than when compiled with OS/VS COBOL or VS COBOL II.

Decide which conversion tools to use and install them

If you use the available conversion tools, you will find that upgrading can be a very simple procedure. The following conversion tools can help in upgrading your source programs to IBM COBOL programs:

COBOL Conversion Tool (CCCA)
The COBOL and CICS/VS Command Level Conversion Aid (CCCA) is **not** for CICS only; it converts any old COBOL to IBM COBOL. The CCCA provides you with either a report of the statements that need to be changed or the actual converted program itself. CCCA is product number 5648-B05.

OS/VS COBOL MIGR compiler option
The MIGR option lists source statements that need to be converted to compile under IBM COBOL NOCMPR2.

CMPR2, FLAGMIG, and NOCOMPILE compiler options
The IBM COBOL CMPR2, FLAGMIG, and NOCOMPILE options list source statements that need to be converted to compile under IBM COBOL NOCMPR2.

Other conversion tools you might want to use include:

- CICS Application Migration Aid (CAMA)—helps convert CICS macro-level code to command-level code. CAMA is product number 5695-061.
- COBOL Structuring Facility (COBOL/SF)—helps structure your COBOL code. COBOL/SF is product number 5696-737.
- COBOL Report Writer Precompiler—enables you to either continue using Report Writer code or convert your Report Writer code to non-Report Writer code.

  The Report Writer Precompiler is product number 5798-DYR. A workstation-based version of the Report Writer Precompiler is also orderable as an optional feature (separately orderable) of VisualAge for COBOL.

These conversion tools are fully described in Appendix C, “Conversion tools for source programs” on page 199.

If you plan to use CCCA, CAMA, COBOL/SF, or COBOL Report Writer Precompiler, install it at this time. For installation instructions, see the documentation for the conversion tool(s) you plan to use.

Educate your programmers on new compiler features

Early in the conversion effort, ensure that your application programmers are familiar with the features of IBM COBOL and the relationship and interdependencies between IBM COBOL, Language Environment, and the Debug Tool and any other application productivity tools your shop uses.

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 Language Environment condition handling and Language Environment callable services.

For information on IBM COBOL and Language Environment education available through IBM, you can call 1-800-IBM-TEACH. You can also get information directly

from Language Environment publications or technical conferences such as GUIDE, SHARE, or the IBM Technical Interchange.

Once your programmers are familiar with IBM COBOL features, they can assist you in taking the inventory of programs as described in "Take an inventory of your applications."

Take an inventory of your applications

In planning the upgrade to IBM COBOL, you need to take a comprehensive inventory of applications in which you have programs that you intend to compile with IBM COBOL. By taking an inventory of your applications, you get a detailed picture of the work that is required. You need to take an inventory of:

- Vendor tools, packages, and products
- COBOL applications

The Edge Portfolio Analyzer can aid in taking an inventory of your existing load modules, see "The Edge Portfolio Analyzer" on page 208 for more information.

Take an inventory of vendor tools, packages, and products

Before you can begin upgrading your source, you need to know if your vendor tools, packages, and products are designed to work with IBM COBOL. Verify that:

- COBOL code generators generate COBOL 85 Standard programs that can be compiled with IBM COBOL.
- COBOL packages are written in COBOL 85 Standard language that can be compiled with IBM COBOL.

Take an inventory of COBOL applications

For each program in your COBOL applications, include at least the following information in your inventory:

VS COBOL II and OS/VS COBOL:

- Programmer responsible
- COBOL Standard level of source program (68, 74, 85)
- Compiler used (ANS COBOL V4, OS/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
- Programs containing Report Writer statements

For OS/VS COBOL only:

- Determine which programs use the following features that are not supported by IBM COBOL:
  - ISAM files
  - BDAM files
  - Communications feature
- Determine which programs use features that might require the purchase of other products:
  - Report Writer statements require the Report Writer Precompiler
- Determine which programs use features that might have different results under IBM COBOL:
  - 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, “Prioritize your applications.”

Prioritize your applications

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

1. Assign complexity ratings to each item in your completed inventory and determine each program or application's resulting overall complexity rating.
2. Determine the conversion priority of each program or application.

Assign complexity ratings

Complexity ratings are defined based on the effort required to convert, test, and coordinate a construct or program. The ratings used in Figure 16 on page 35 are defined as:

0 All code converted by CCCA without error; code compiles correctly under IBM COBOL

1-3 Requires moderate testing
Requires moderate coordination
Most code converted without error by CCCA

4 Requires CCCA and possible manual conversion
Requires special testing considerations

5-6 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)

7-8 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

Based on the complexity ratings shown above (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 16 shows estimated complexity ratings for conversions of specific program attributes.

Figure 16. Complexity ratings for programs attribute conversions

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		
Fixed file attribute mismatch (FS 39)[4]		4		
COBOL 74 Standard COPY library members		1	M	C
ANS COBOL V4 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	2	M	C
	6 +	3	M	C
No source code for module	Module needs rewrite	10[1]		
	Module does not need to be upgraded	6		
CICS macro level program		10[2]		
Compiled by Full ANS COBOL V4 compiler (pre-OS/VS COBOL compiler)		4		C
Compiled by OS/VS COBOL Release 2 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
	BDAM	10		C[3]
	TCAM	10		
Uses Report Writer language (if not using Report Writer Precompiler)		6	M	C
Uses Report Writer language (if using Report Writer Precompiler)		0		
CICS		4		
Uses language with changed results	Complex OCCURS DEPENDING ON	4		C
	Combined abbreviated relation conditions	6	M	
	Floating-point arithmetic	6	M	
	Exponentiation	6	M	
	Signed data	2		
	Binary data	2		

Notes to Complexity Ratings Table:

1 Non-IBM vendors can recreate COBOL source code from object code.

2 You can use the CICS Application Migration Aid to help convert CICS macro-level programs to command-level programs.

3 This is a partial conversion.

4 For additional information, see Appendix I, "Preventing file status 39 for QSAM files" on page 241.

On categories marked **M** you can gather information using the OS/VS COBOL MIGR option. On categories marked **C** you can gather information using the COBOL conversion tool (CCCA). You can also use the IBM COBOL FLAGMIG compiler option to identify elements that need to be changed.

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

Figure 17. Assigning program conversion priorities

Conversion priority	Complexity rating	Other considerations
1	0 to 3	Great importance to your organization Low conversion effort using conversion tools
2	4 to 6	Great importance to your organization Medium conversion effort using conversion tools
	0 to 3	Medium importance to your organization Low conversion effort using conversion tools
3	7 to 8	Great importance to your organization High conversion effort using conversion tools
	3 to 6	Medium importance to your organization Medium conversion effort using conversion tools
	0 to 3	Small importance to your organization Low conversion effort using conversion tools
4	9 to 10	Great importance to your organization Very high conversion effort
	7 to 8	Medium importance to your organization High conversion effort using conversion tools
	3 to 6	Small importance to your organization Medium conversion effort using conversion tools
5	9 to 10	Medium importance to your organization Very high conversion effort
	7 to 8	Small importance to your organization High conversion effort using conversion tools
6	9 to 10	Small importance to your organization very high conversion effort

Consider the following when deciding on conversion priorities:

- If your application is at the limits of the storage available below the 16M line, it is a prime candidate for conversion to IBM COBOL. With OS/390, MVS, or VM/ESA architecture you can obtain Virtual Storage Constraint Relief.
- If the program cannot run under Language Environment, you must convert it. For example, an OS/VS COBOL program that CALLs or is called by a PL/I program must be upgraded to IBM COBOL.

Once you determine the priority of each program you need to upgrade and the effort required to upgrade those programs, you can decide the order in which you want to convert your applications and programs.

Set up upgrade/no upgrade 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 IBM COBOL.

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 under Language Environment.
- Programs of low importance to your organization that run correctly under Language Environment and that would take a very high conversion effort.
- Programs that are being phased out of production.

Note, however, that there might be restrictions on running existing modules mixed with upgraded programs. See Chapter 16, “Adding IBM COBOL programs” on page 170.

Set up a conversion procedure

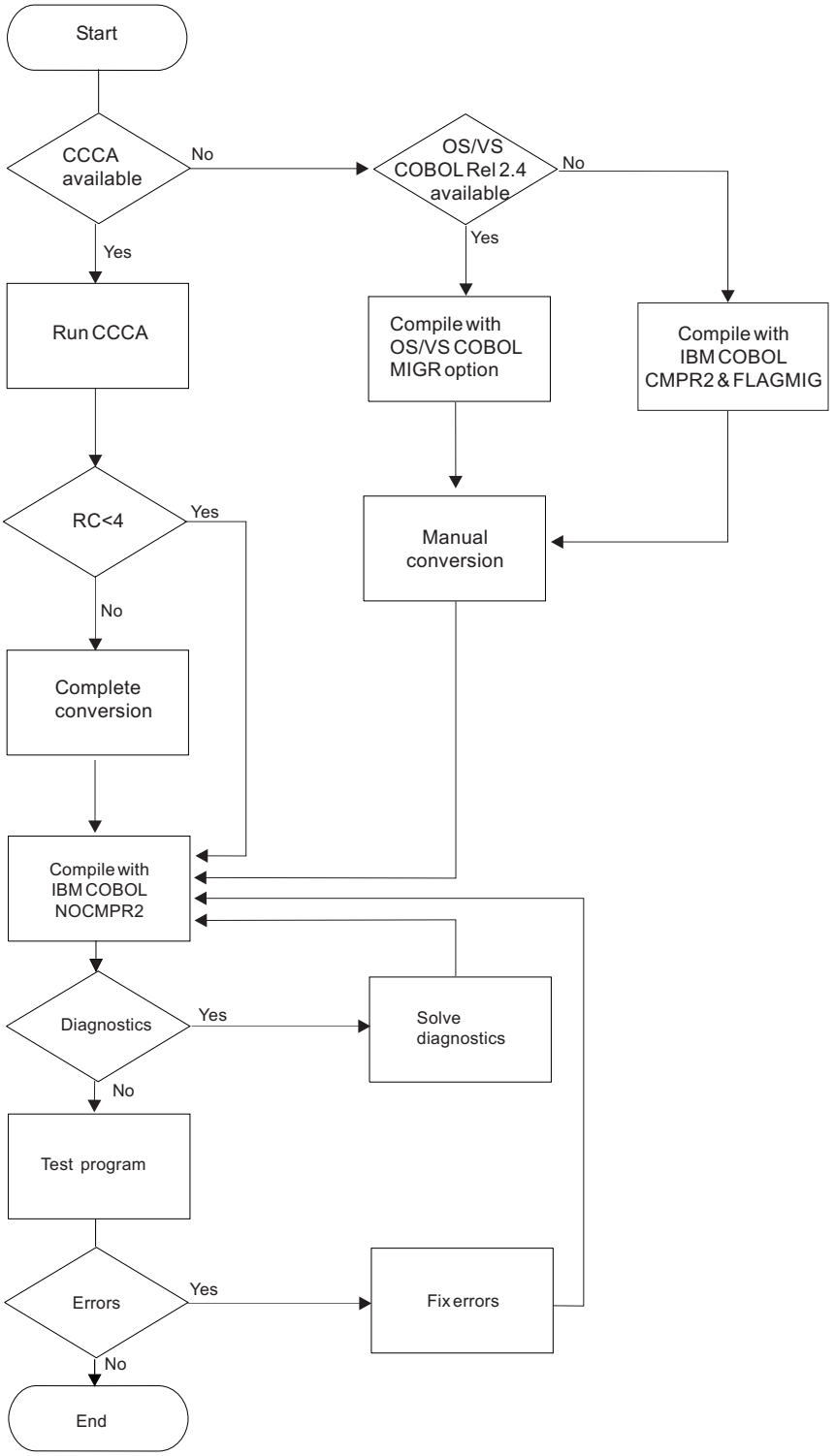
The summaries and diagrams on the following pages outline the steps required to upgrade five types of programs:

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

In the following flowcharts, you are directed to manually upgrade your programs if you are not using CCCA. If you do not want to use CCCA, you should consider using a non-IBM vendor's conversion tool before attempting a manual conversion.

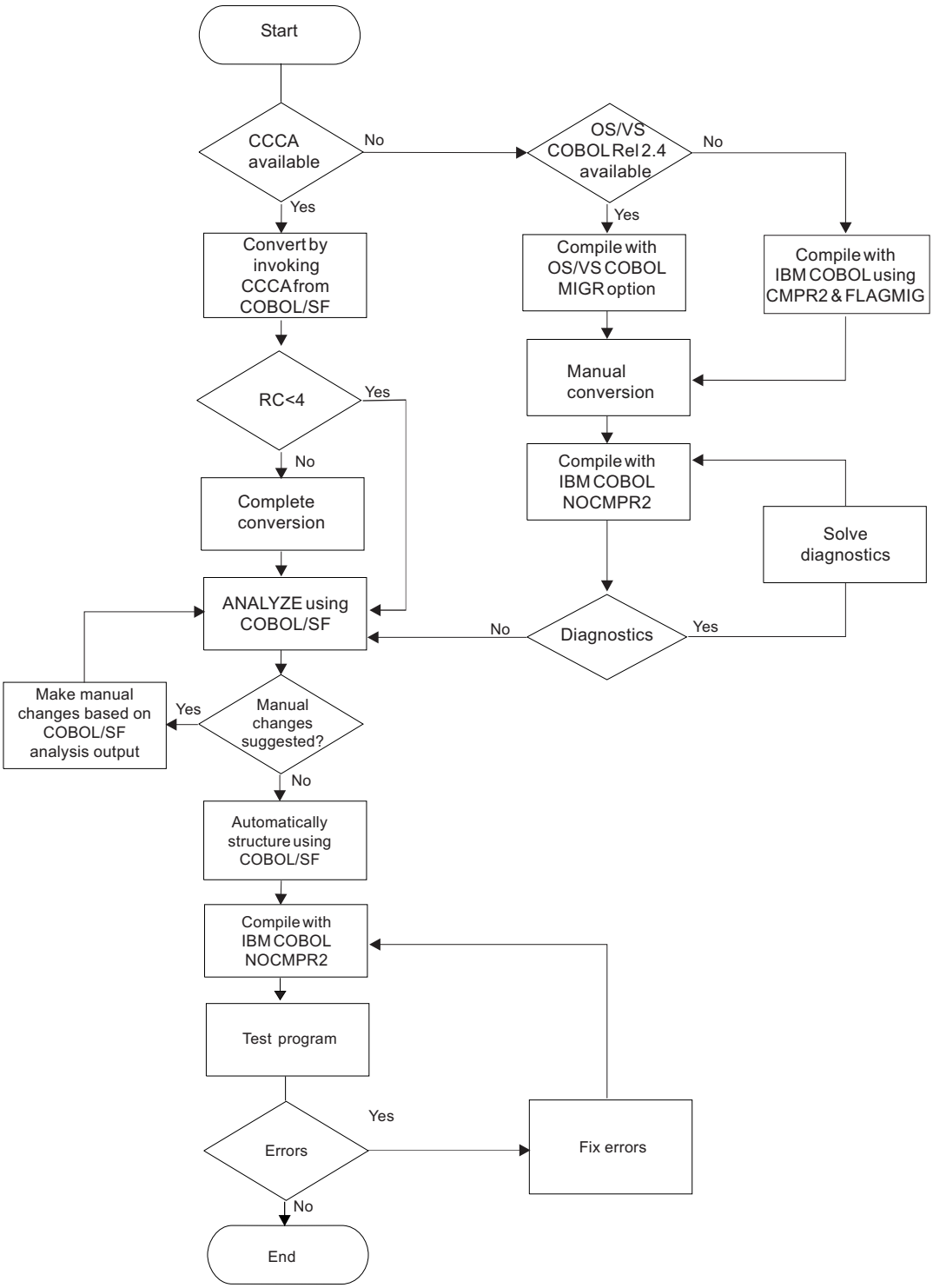
Programs without CICS or Report Writer

To convert an OS/VS COBOL program that contains neither CICS commands nor Report Writer statements to an IBM COBOL program, do the following:



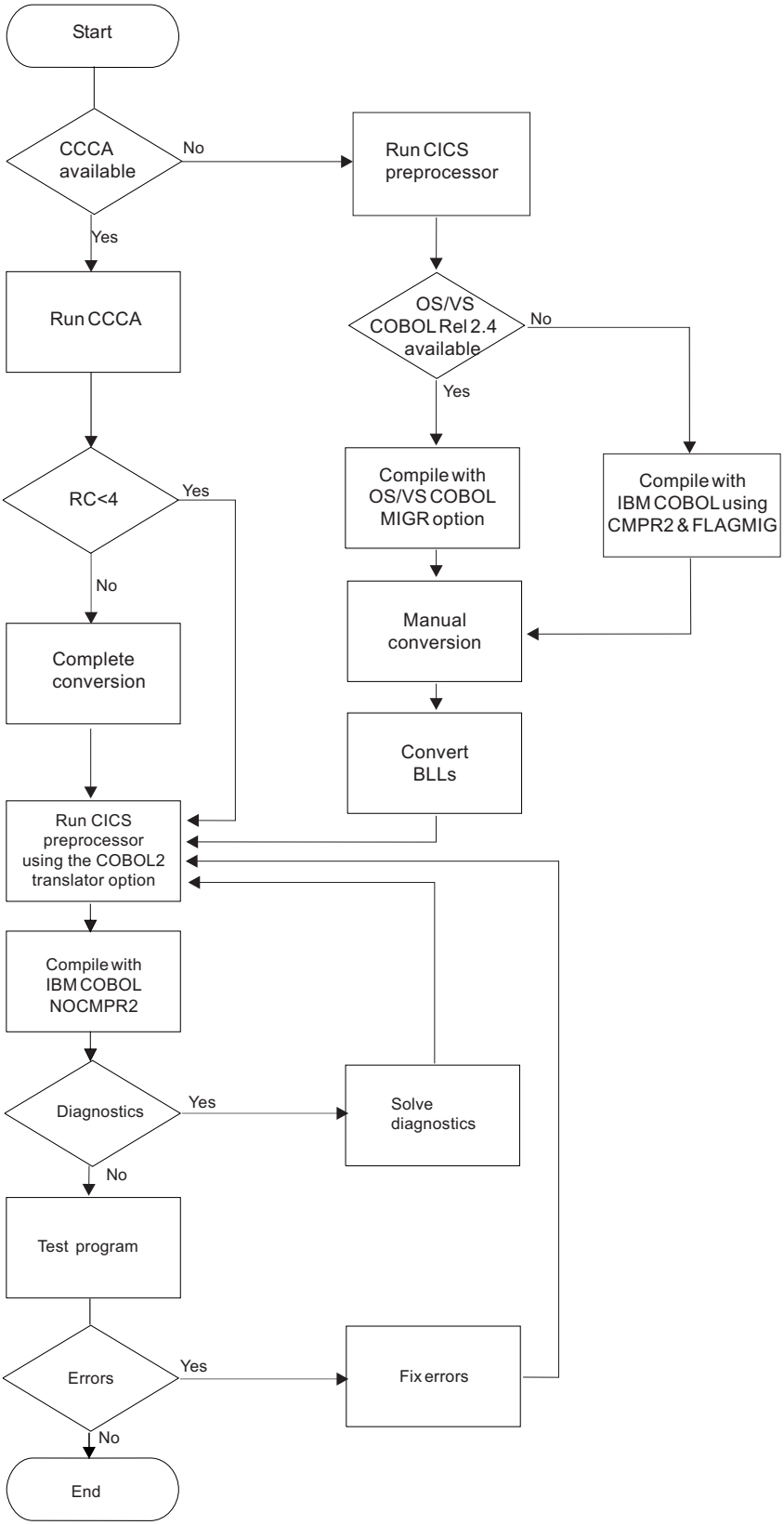
Programs converted to structured programming code

To convert a non-CICS OS/VS COBOL program, which does not contain Report Writer statements, to IBM COBOL and then to a structured format (this requires the availability of COBOL/SF), do the following:



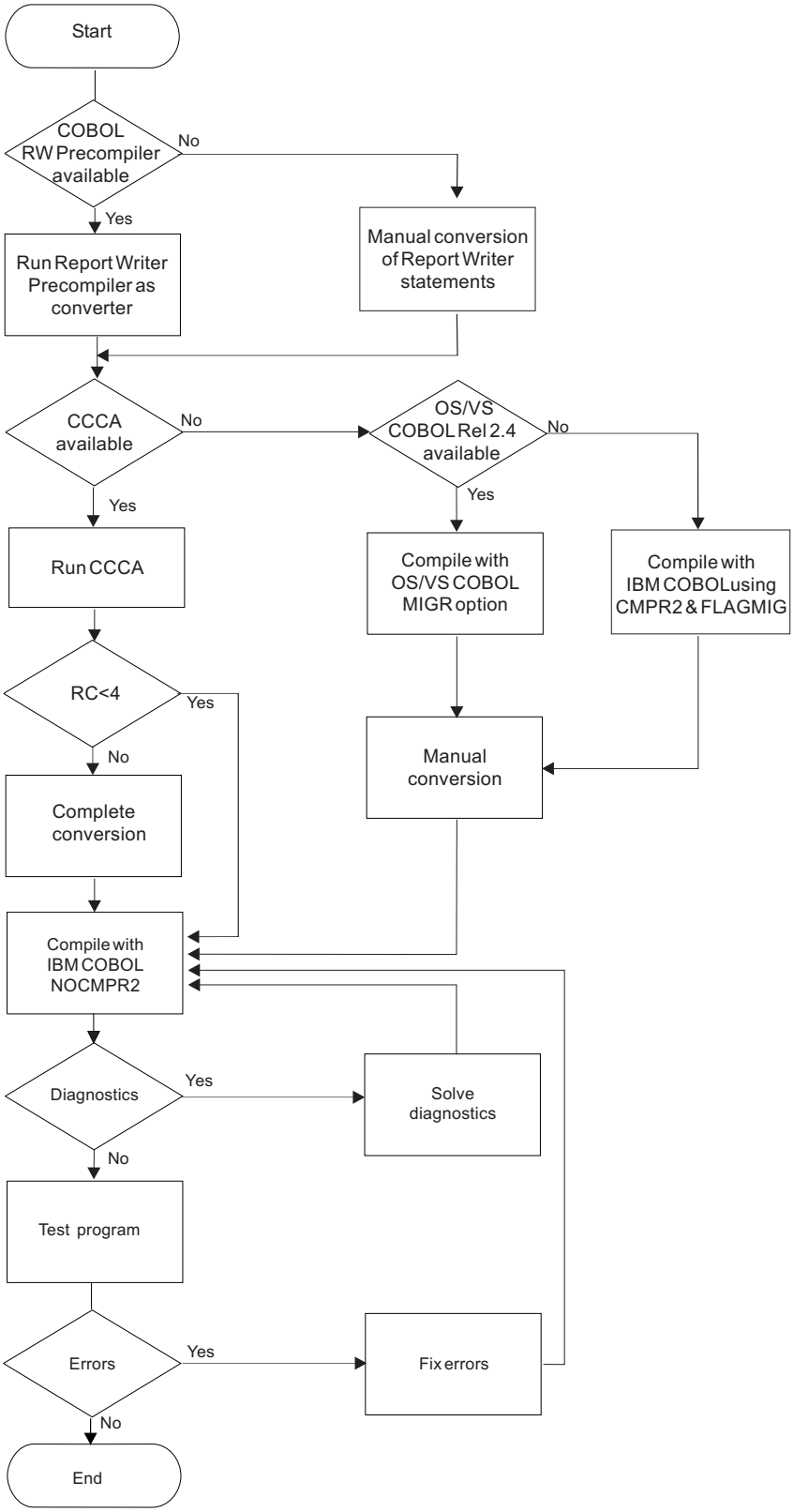
Programs with CICS

To convert an OS/VS COBOL program that contains CICS commands to a IBM COBOL program, do the following:



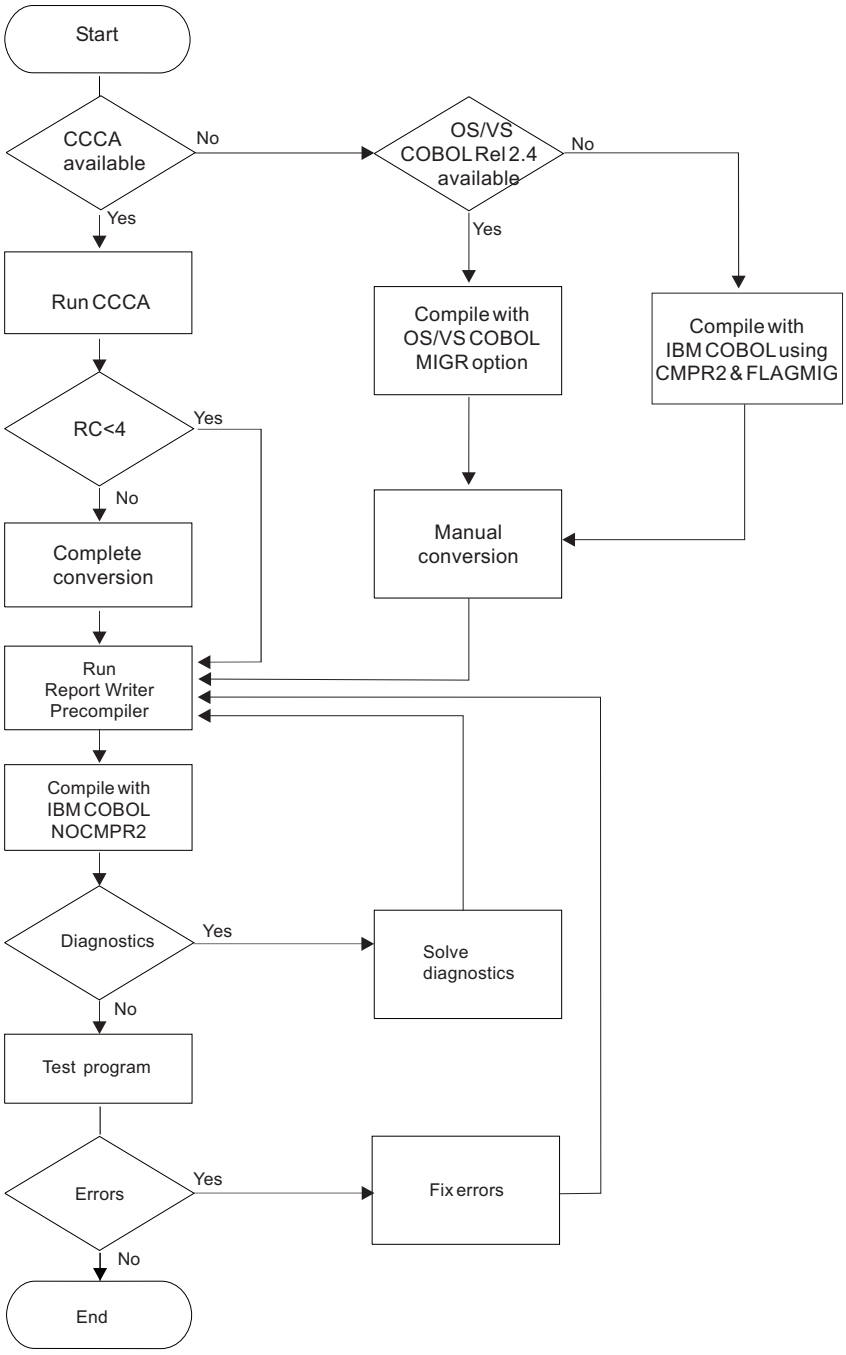
Programs with Report Writer statements to be discarded

To convert an OS/VS COBOL program that contains Report Writer statements to an IBM COBOL program, and remove all Report Writer statements, do the following:



Programs with Report Writer statements to be retained

To convert an OS/VS COBOL program that contains Report Writer statements to an IBM COBOL program, and retain the Report Writer statements in the source code, do the following:



Make application program updates

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

Save the existing source as a back up—a benchmark to compare to and a version to recover to—if the converted modules have problems.

1. Update job and module documentation

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

2. Update available source code

   Whenever possible, use the conversion tools described in Appendix C, “Conversion tools for source programs” on page 199. Otherwise, update the source code manually.

3. Compile, link-edit, and run

   Once the source is updated, you can process the program as you would a newly written IBM COBOL program. (Note, you need the Language Environment run time installed.)

4. Debug

   Analyze program output and, if the results are not correct, use the Debug Tool or Language Environment dump output to uncover any errors present.

5. Test the converted programs

   After upgrading your source to IBM COBOL, set up a procedure for regression testing. Regression testing will help to identify:

   - Fixed file attribute mismatches (file status 39 problems). Verify that your COBOL record descriptions, JCL DD statements, and physical file attributes match. For more information, see Appendix I, “Preventing file status 39 for QSAM files” on page 241.
   - Dependency on WORKING-STORAGE being initialized to binary zeros. If you have WORKING-STORAGE zero dependency, specify the Language Environment STORAGE(00) run-time option.
   - Performance differences.
   - Sign handling problems—S0C7 abends. The data's sign must match the signs allowed by NUMPROC compiler option suboption you specify.
   - DATA(24) issues. Do not mix AMODE(24) programs with 31-bit data.

   Once you have established a regression testing procedure, and once your programs run correctly, test them against a variety of data:

   a. Locally—each program separately
   b. 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.

6. Repeat when necessary

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

7. Cut over to production mode

   When your testing shows the entire application receives the expected results, you can move the entire unit over to production mode. (This assumes your production system is already using the Language Environment run time. If not, STEPLIB to the Language Environment run time. See “Decide how to phase Language Environment into production mode” on page 26.)

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

   - Under OS/390, MVS, and VM/ESA, run the old version as a substitute from the latest productivity checkpoint.
   - Under DB2 and IMS return to the last commit point and then continue processing from that point using the unmigrated COBOL program. (For DB2, use an SQL ROLLBACK WORK statement.)
   - For non-CICS applications, use your shop's backup and restore facilities to recover.

8. Run in production mode

   After cut over, monitor the application for a short time to ensure you are getting the results expected. After that, your source conversion task is completed.

Moving existing applications to Language Environment

Chapter 5. Running existing applications under Language Environment

Depending on the characteristics of your applications, you might need to make application modifications and perform some of the following Language Environment customization tasks to ensure that your current applications run under Language Environment:

- Set recommended default Language Environment run-time options
- Invoke existing applications
- Link-edit existing applications
- Obtain a system dump or a CICS transaction dump
- Get compatible abend behavior
- Ensure return code value compatibility

Other factors also apply to ensure compatibility, depending on if you are moving your run time from OS/VS COBOL or VS COBOL II. For details, see:

- Chapter 6, “Moving from the OS/VS COBOL run time” on page 58
- Chapter 7, “Moving from the VS COBOL II run time” on page 70

Set recommended default Language Environment run-time options

The Language Environment IBM-supplied default run-time option settings might not provide the same run-time behavior as the OS/VS COBOL or VS COBOL II run-time options. Because there are differences, 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.

Recommended run-time options for non-CICS applications

Figure 18 describes the Language Environment run-time options that are highly recommended for existing non-CICS COBOL applications. For a complete list of Language Environment run-time options, see *Language Environment Programming Reference*.

Figure 18 (Page 1 of 3). Recommended Language Environment run-time options for non-CICS COBOL applications

Option	LanEnv default setting	Recommended COBOL setting	Comments
ABTERMENC	RETCODE (all releases up to and including V2R8). ABEND (V2R9 or later)	ABEND	ABTERMENC(ABEND) ensures you receive abend codes similar to those issued by VS COBOL II or OS/VS COBOL when an abend, program check, or severe error occurs. For additional VS COBOL II run-time messages considerations, see “Abend codes” on page 83. ABTERMENC(ABEND) ensures that your application ends with an abend; to obtain a system dump, see “Obtain a system dump or a CICS transaction dump” on page 53.
CBLOPTS	ON	ON	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 only affects applications with COBOL as the main program.

Figure 18 (Page 2 of 3). Recommended Language Environment run-time options for non-CICS COBOL applications

Option	LanEnv default setting	Recommended COBOL setting	Comments
CBLQDA	ON	OFF	CBLQDA(OFF) suppresses QSAM dynamic allocation for files (in programs compiled with NOCMPR2) not available when OPEN OUTPUT, OPEN I-O (optional file), or OPEN EXTEND (optional file) statements are directed to a QSAM file. This behavior is compatible with programs compiled with OS/VS COBOL, VS COBOL II Release 2, and VS COBOL II Release 3 and later compiled with CMPR2. CBLQDA(ON) conforms to the COBOL 85 Standard.
RTEREUS	OFF	OFF	RTEREUS is not recommended as an installation default. If you do use RTEREUS, use it for specific applications only and make sure that you understand the possible side effects and restrictions, for example: • RTEREUS(ON) is ignored if CEEPIPI or DB2 stored procedures are used. • Under Language Environment, RTEREUS(ON) is only supported in a single enclave environment unless you modify the behavior using the IGZERREO CSECT. With the IBM-supplied default setting for COBOL's reusable environment, applications that attempt to create nested enclaves will terminate with error message IGZ0168S. Nested enclaves can be created by applications that use SVC LINK or CMSCALL to invoke applications programs. One example is when an SVC LINK is used to invoke an application program under ISPF that is using ISPF services (such as CALL 'ISPLINK' and ISPF SELECT). • If a Language Environment reusable environment is established (using RTEREUS), attempts to run a C or PL/I main program under Language Environment will fail. For example, when running on ISPF with RTEREUS(ON): – The first program invoked by ISPF is a COBOL program. A Language Environment reusable environment is established. – At some other point, ISPF invokes a PL/I or C program. The initialization of the PL/I or C program will fail. • If a large number of COBOL programs are run under the same MVS task, you can get out of region abends. This is because all storage acquired by Language Environment to run COBOL programs is kept in storage until the MVS task ends or the Language Environment environment is terminated. • Language Environment termination will not be driven unless a STOP RUN is executed to end the enclave. As a result: – Language Environment storage and run-time options reports are not produced by Language Environment. – COBOL files that were not closed by the COBOL program will not be closed by Language Environment, which can result in unpredictable data in the file (for example, records not being written, or the last record being written twice).
ANYHEAP BELOWHEAP HEAP LIBSTACK STACK			These run-time options help you manage storage. For **STACK**, if any program in your application is running AMODE(24), specify the suboption BELOW (and also specify ALL31(OFF)). If all of the programs in your application are AMODE(31), specify the suboption ABOVE (and also specify ALL31(ON)). On COBOL, the recommended setting for STACK is 64K,64K,BELOW,KEEP.
TERMTHDACT	TRACE	UADUMP or UATRACE	Use TERMTHDACT(UADUMP), TERMTHADACT(UATRACE), or TERMTHDACT(UAONLY) to receive a system dump under Language Environment when the environment is terminating due to a severe error (for example, a program check or abend). UATRACE and UAONLY are available starting with OS/390 Version 2 Release 7. Alternatively, use an abnormal termination exit. See “Obtain a system dump or a CICS transaction dump” on page 53 for details.

Figure 18 (Page 3 of 3). Recommended Language Environment run-time options for non-CICS COBOL applications

Option	LanEnv default setting	Recommended COBOL setting	Comments
TRAP	ON	ON	TRAP specifies how Language Environment routines handle abends and program checks. In order for applications to run successfully, you must specify TRAP(ON). TRAP(ON) also enables Language Environment to support the existing condition handling mechanisms provided by both VS COBOL II (STAE run-time option) and OS/VS COBOL (STATE, FLOW, COUNT, and SYMDMP debugging options).

Other run-time options affecting non-CICS applications

The following run-time options can also determine whether existing applications run under Language Environment and provide expected results. No specific suboption can be recommended since the default setting is dependent on each individual shop's needs.

ALL31

ALL31(OFF) is required for applications with AMODE(24) programs, such as:

- OS/VS COBOL programs
- VS COBOL II NORES programs
- Other AMODE(24) non-COBOL programs

ALL31(ON) allows EXTERNAL data to be allocated anywhere within the 31-bit addressing range and improves run-time performance. The IBM-supplied default for non-CICS is ALL31(OFF).

MSGFILE

When you use the Language Environment run-time option MSGFILE(ddname) to specify the destination of messages, there are restrictions on the names that you can use for the ddname of the output message file. Do not use any of the following ddnames:

SYSABEND	SYSCOUNT	SYSDBOUT	SYSDTERM
SYSIN	SYSLIB	SYSLIN	
SYSPUNCH	SYSUDUMP	SYSLOUT	

The IBM-supplied default is MSGFILE(SYSOUT).

STORAGE

For programs that depend on WORKING-STORAGE that is initialized to binary zeros, this option can be used to initialize storage acquired by a program compiled with RENT to binary zeros, except when VALUE clauses are specified. It can also be used to set all the EXTERNAL data records of a program to binary zeros. However, to improve performance, you should explicitly initialize only those data items that require initialization.

WSCLEAR and STORAGE(00) do not affect programs compiled with NORENT.

To receive the Language Environment-equivalent of the VS COBOL II WSCLEAR run-time option, set the first suboption of the Language Environment STORAGE run-time option to 00. For example, STORAGE(00,NONE,NONE,8K).

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

Recommended run-time options for CICS applications

Figure 19 describes the Language Environment run-time options that are highly recommended for existing COBOL CICS applications. On CICS, Language Environment has different default settings than under non-CICS (most of the Language Environment default settings are the same as the recommended COBOL settings).

For a complete list of Language Environment run-time options, see the *Language Environment Programming Reference*.

Figure 19 (Page 1 of 2). Recommended Language Environment run-time options for COBOL CICS applications

Option	LanEnv default setting on CICS	Recommended COBOL setting	Comments
ABTERMENC	ABEND	ABEND	Ensures you receive abend codes similar to those issued by VS COBOL II or OS/VS COBOL when an abend, program check, or severe error occurs. ABTERMENC(ABEND) ensures that you get a system abend code; to get a system dump, see "Obtain a system dump or a CICS transaction dump" on page 53.
ALL31	ON	ON	ALL31(ON) allows Language Environment to allocate its control blocks above the line. With ALL31(OFF), Language Environment increases its use of below-the-line storage. For more information on the differences in Language Environment storage usage on CICS for ALL31(OFF) and ALL31(ON), see "Virtual storage requirements" on page 19. ALL31(ON) is the recommended setting for COBOL applications. You can use ALL31(ON) if all of your VS COBOL II and IBM COBOL programs are AMODE 31, even if you are running OS/VS COBOL programs in your CICS regions. (Load modules containing only OS/VS COBOL and assembler programs are AMODE(24) and are not affected by the setting of ALL31.) **Note:** To run with ALL31(ON), every program in the load module must be 31-bit enabled, including assembler programs. Use ALL31(OFF) if your load modules contain VS COBOL II, COBOL/370, or IBM COBOL programs and also contain assembler programs that require AMODE(24).
ANYHEAP BELOWHEAP HEAP LIBSTACK STACK	See *Language Environment Installation and Customization*.	If using a release of Language Environment prior to Version 2 Release 7, change LIBSTACK to LIBSTACK(1K,1K,FREE); otherwise use the default settings.	Language Environment provides these run-time options to help manage storage.

Figure 19 (Page 2 of 2). Recommended Language Environment run-time options for COBOL CICS applications

Option	LanEnv default setting on CICS	Recommended COBOL setting	Comments
ANYHEAP BELOWHEAP HEAP LIBSTACK STACK		See Language Environment Installation and Customization	Language Environment provides these run-time options to help manage storage.
		If using a release of Language Environment prior to Version 2 Release 7, change LIBSTACK to LIBSTACK (1K,1K,FREE); otherwise use the default settings.	
TERMTHDACT	TRACE	UADUMP or UATRACE	Use TERMTHDACT(UADUMP), TERMTHADACT(UATRACE), or TERMTHDACT(UAONLY) to receive a system dump under Language Environment when the environment is terminating due to a severe error (for example, a program check or abend). UATRACE and UAONLY are available starting with OS/390 Version 2 Release 7. Alternatively, use an abnormal termination exit. See “Obtain a system dump or a CICS transaction dump” on page 53 for details.
TRAP	ON	ON	TRAP specifies how Language Environment routines handle abends and program interrupts. In order for applications to run successfully, you must specify TRAP(ON).

Other run-time options affecting CICS applications

The following run-time options can also determine whether existing applications run under Language Environment and provide expected results. No specific suboption can be recommended since the default setting is dependent on each individual shop's needs.

CBLPSHPOP

CBLPSHPOP is used to control whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a VS COBOL II or IBM COBOL subroutine is called using the COBOL CALL statement.

CBLPSHPOP(ON) ensures you receive behavior that is compatible with the behavior of running VS COBOL II programs with the VS COBOL II run time. CBLPSHPOP(OFF) can yield performance benefits. For details, see “CICS HANDLE commands and the CBLPSHPOP run-time option” on page 96.

STORAGE

For programs that depend on WORKING-STORAGE that is initialized to binary zeros, this option can be used to initialize storage acquired by a program compiled with RENT to binary zeros, except when VALUE clauses are specified. It can also be used to set all the EXTERNAL data records of a program to binary zeros. However, to improve performance, you should explicitly initialize only those data items that require initialization.

To receive the Language Environment-equivalent of the VS COBOL II WSCLEAR run-time option, set the first suboption of the Language Envi-

ronment STORAGE run-time option to 00. For example, STORAGE(00,NONE,NONE,0K).

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

Invoke existing applications

To access Language Environment 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.

Note: Make sure your program names do not begin with AFH, CEE, EDC, IBM, IGZ, ILB, or FOR. These prefixes are reserved for Language Environment library routine module names.

For non-CICS applications

The following sections detail the changes required for non-CICS applications. For considerations for programs run on IMS or CMS, see Appendix M, "IMS considerations" on page 282. For more information on how to prepare and run your programs with Language Environment, see the *Language Environment Programming Guide*.

Specify the correct library: To invoke existing applications when running under Language Environment, you need to:

Under OS/390 or MVS

Replace your current library with the Language Environment SCEERUN library.

Under CMS

Replace your current library with the SCEERUN LOADLIB library. In addition, to run OS/VS COBOL programs under Language Environment on CMS, specify the SCEEILBO LOADLIB library.

Specify alternate DDNAMES (optional): With Language Environment, you can indicate the destination for Language Environment output by changing the ddname in the MSGFILE run-time option to the ddname you want. Figure 20 lists the default ddnames for Language Environment output.

Figure 20. Specification of new DDNAMEs

Output	Default ddname	Dynamically allocated
Messages	SYSOUT	Yes
Run-time options report (RPTOPTS)	SYSOUT	Yes
Storage reports (RPTSTG)	SYSOUT	Yes
Dumps	CEEDUMP	Yes

You do not need to alter your JCL, CLISTs, or Rexx EXECs to define the ddnames for Language Environment messages, reports, or dumps **unless** the defaults used by Language Environment do not meet the needs of your shop. The Language Environment default destinations are:

- On OS/390 and MVS: SYSOUT=*
- On CMS: FILEDEF SYSOUT TERMINAL

- On TSO: ALLOC DD(SYSOUT) DA(*)

Remove DDNAMES no longer required (optional): The following ddnames are not required when running with Language Environment:

- SYSABOUT (used by VS COBOL II run time only)
- SYSDBIN (used by VS COBOL II run time only)
- SYSDBOUT

You are not required to remove these ddnames. This information is provided in case you want to eliminate unnecessary coding in your JCL, CLISTs, or Rexx EXECs.

For CICS applications

To run Language Environment on CICS, you need to perform several required steps. For details on how to invoke COBOL applications running on CICS under Language Environment, see:

- For MVS, *Language Environment Installation and Customization*
- For OS/390, *Language Environment for OS/390 Customization*

Under CICS, Language Environment 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 21 lists the types of Language Environment output and location.

Figure 21. Location of Language Environment output under CICS

Output	Transient data queue
Messages	CESE
Run-time options report (RPTOPTS)	CESE
Storage reports (RPTSTG)	CESE
Dumps	CESE
DISPLAY UPON SYSOUT output	CESE

Link-edit existing applications

After determining which of your existing applications either require or will benefit from link-editing with Language Environment, you need to specify the correct library name. The Language Environment link-edit library is the same for non-CICS applications as for CICS applications.

Under OS/390 and MVS

Include the Language Environment SCEELKED in the SYSLIB concatenation.

Under CMS

Include SCEELKED in the GLOBAL TXTLIB command.

Note: If you link-edit with the NCAL linkage editor option, ensure that all of the required run-time routines from SCEELKED are included in the load module. Otherwise, unpredictable errors will occur (typically a program check).

There are some names in the SCEELKED library that do not follow IBM naming conventions, and that can conflict with your subprogram names. For example, if you have a statically called subroutine named DUMP and if SCEELKED is ahead of your private subroutine library in the concatenation at link-edit time, then your references to DUMP will be resolved in SCEELKED. In this example, the FORTRAN routine AFHUDUMS will be link-edited in, and you could get incorrect results, loss of function, or slower performance as a result. (Another common name is ABORT, which is an entry point in EDC4$05C, a C run-time library routine.)

There are a couple of ways to avoid these problems:

- You can check the names in the SCEELKED data set against the names of your private subroutines. If there are any duplicates, you can rename your private subroutines so that they do not have the same names as the names in the SCEELKED data set.
- Another way is to place your private subroutine libraries before SCEELKED in the SYSLIB concatenation. However, doing this could result in losing function that is available under Language Environment if your application contains Fortran or C/C++ programs. Changing the name of your subroutine to avoid the conflict with the Language Environment subroutine is preferable to placing you private subroutine libraries ahead of SCEELKED.

To determine which applications require link-edit with Language Environment, see either of the following sections:

- "Determining which programs require link-edit" on page 59, if running under the OS/VS COBOL run time
- "Determining which programs require link-edit" on page 71, if running under the VS COBOL II run time

Obtain a system dump or a CICS transaction dump

To receive a system dump or a CICS transaction dump under Language Environment when the environment is terminating due to a severe error (for example, a program check or abend) you have two options. The option you choose depends on how much diagnostic information you want Language Environment to produce.

Both options are affected by the Language Environment TERMTHDACT run-time option, which sets the level of diagnostic information produced by Language Environment when the environment is terminating due to a severe error. For details on the TERMTHDACT run-time option, see the *Language Environment Programming Reference*.

Method 1: Specify the TERMTHDACT run-time option

Language Environment provides three TERMTHDACT suboptions (UADUMP, UATRACE, and UAONLY) to produce different types of Language Environment dumps.

TERMTHDACT(UADUMP) causes Language Environment to produce a Language Environment formatted dump plus a system dump. With the TERMTHDACT(UADUMP) setting, the Language Environment dump includes a traceback and a dump of the thread/enclave/process level storage and control blocks.

The UADUMP option is available starting with Language Environment Release 5.

TERMTHDACT(UATRACE) causes Language Environment to generate a message indicating the cause of the termination, a trace of the active routines on the activation stack, and a U4039 abend which allows a system dump of the user address space to be generated.

TERMTHDACT(UAONLY) causes Language Environment to generate a U4039 abend which allows a system dump of the user address space to be generated. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space. Under CICS, you will get a CICS transaction dump.

UATRACE and UAONLY are available starting with OS/390 Version 2 Release 7.

Method 2: Specify an abnormal termination exit

Using an abnormal termination exit is the recommended approach to getting a system dump or a CICS transaction dump when you want to set the TERMTHDACT run-time option to a value other than DUMP or UADUMP.

When you specify an abnormal termination exit, you can get a system dump or a CICS transaction dump before Language Environment frees the resources it has acquired; thus reducing the amount of diagnostic information you receive in the formatted dump, without impacting the system dump.

Abnormal termination exit under non-CICS

For non-CICS applications, the sample abnormal termination exit is SAMPDAT1, as shown in Figure 22 on page 55. SAMPDAT1 provides system abend dumps.

Abnormal termination exit under CICS

For CICS applications, the sample abnormal termination exit is SAMPDAT2, as shown in Figure 23 on page 56. SAMPDAT2 provides transaction dumps.

For information on using an abnormal termination exit, see:

- On OS/390: *Language Environment for OS/390 Customization*
- On MVS: *Language Environment Installation and Customization*

Abnormal termination exit (non-CICS)

```
*********************************************************************
*                                                                   *
*  Do a system DUMP whenever an unhandled condition occurs.         *
*                                                                   *
*********************************************************************
SAMPDAT1 CEEENTRY PPA=ASMPPA,MAIN=NO
         L     2,0(,1)                   Put the pointer to the CIB
*                                        address in R2.
         L     2,0(,2)                   Put the CIB address in R2.
*********************************************************************
* Set up the ESTAE and force the abend with a dump.
*********************************************************************
         ESTAE ESHDLR
         ABEND 4039,REASON=0,DUMP        FORCE DUMP
RETRY    ESTAE 0
         CEETERM                         All done, return to LE/370
         DROP  11,13
         USING *,15
ESHDLR   STM   14,12,12(13)
NEXT     L     11,MODENT
         USING SAMPDAT1,11
         DROP  15
         SETRP RC=4,RETADDR=RETRY,RETREGS=YES,FRESDWA=YES
         LM    14,12,12(13)
         BR    14
MODENT   DC    A(SAMPDAT1)
ASMPPA   CEEPPA
         CEEDSA
         CEECAA
SDWA     IHASDWA
         END SAMPDAT1
```

Figure 22. Non-CICS abnormal termination exit sample

Abnormal termination exit (CICS)

```
*ASM CICS(NOPROLOG NOEPILOG NOEDF SYSEIB)
*********************************************************************
*                                                                   *
*  Do a transaction DUMP whenever an unhandled condition occurs.    *
*                                                                   *
*********************************************************************
SAMPDAT2 CEEENTRY PPA=ASMPPA,MAIN=NO,AUTO=STORLEN
         USING DFHEISTG,DFHEIPLR
*********************************************************************
* Ask CICS to produce a transaction dump.
*********************************************************************
         EXEC CICS ADDRESS EIB(DFHEIBR)
         EXEC CICS DUMP TRANSACTION DUMPCODE('4039') TASK NOHANDLE
*********************************************************************
* To see if the dump was successful, add code here to check field EIBRESP.
*********************************************************************
         CEETERM                          All done, return to LE/370
ASMPPA   CEEPPA
         CEEDSA
         CEECAA
         DFHEISTG                         Extended save area for CICS
STORLEN  EQU   *-DFHEISTG
         COPY  DFHEIBLK
         EXTRN DFHEAI
DFHEIPLR EQU   13
DFHEIBR  EQU   10
         END SAMPDAT2
```

Figure 23. CICS abnormal termination exit sample

Get compatible abend behavior

Use the Language Environment ABTERMENC run-time option and the assembler user exit to receive abend behavior similar to OS/VS COBOL and VS COBOL II.

ABTERMENC(ABEND) ensures you receive system abend codes similar to those issued by VS COBOL II or OS/VS COBOL when an abend, program check, or severe error occurs. (To get a system dump, see “Obtain a system dump or a CICS transaction dump” on page 53.)

To get user abend codes similar to what you received under VS COBOL II (where xxx is the IGZ message number) for unhandled Language Environment software generated conditions, do **one** of the following:

- Modify the assembler user exit (CEEBXITA) by copying the code from the COBOL sample assembler user exit (CEEBX05A) into CEEBXITA.
- Use the sample user condition handler CEEWUCHA as described in “Using CEEWUCHA” on page 84.

Ensure return code value compatibility

Language Environment calculates return code values differently than OS/VS COBOL or VS COBOL II. There are two cases when you might receive different return code values when running under Language Environment:

1. If you specify the ABTERMENC(RETCODE) run-time option **or**
2. If you modify the Language Environment assembler user exit to manipulate the return code value

Chapter 6. Moving from the OS/VS COBOL run time

This chapter provides detailed information about running OS/VS COBOL programs under Language Environment. It includes information on:

- Determining which programs require link-edit
- Determining which programs require upgrade
- Comparing run-time options and specification methods
- Closing files in non-COBOL and OS/VS COBOL programs
- Running in a reusable run-time environment
- Managing dump services
- Using ILBOABN0 to force an abend
- Using SORT/MERGE in OS/VS COBOL programs
- Understanding SYSOUT output changes
- Communicating with other languages
- Additional CICS considerations

Additional information on moving your run time to Language Environment is included in:

- Appendix D, “Applications with COBOL and assembler” on page 209
- Appendix M, “IMS considerations” on page 282
- Appendix N, “CMS considerations” on page 287

Each section in this chapter indicates if it is applicable to existing OS/VS COBOL programs compiled with the RES compiler option, NORES compiler option (including whether the programs are link-edited with Language Environment), and for applications running on CICS, by using the following notation:

This section applies to:

✔ RES
✔ NORES
✔ CICS
✔ NORES Linked

RES

An application comprised of programs compiled with RES.

NORES

An application comprised of programs compiled with NORES. The NORES programs have not been link-edited with Language Environment.

OR

An application comprised of OS/VS COBOL programs compiled NORES that **is** link-edited with Language Environment, but **does not** contain any of the following:

- A VS COBOL II program
- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- The IGZCBSN or IGZCBSO run-time routine

CICS

An application that runs on CICS.

NORES linked

An application comprised of programs compiled with NORES. The NORES programs have been link-edited with Language Environment and now behave as if they were RES. The application **does** contain at least one of the following:

- A VS COBOL II program
- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- The IGZCBSN or IGZCBSO run-time routine

Note: For multiple load module applications, if the **first executed** load module contains one of the above, the application will behave as if it were RES.

Determining which programs require link-edit

This section applies to:

✔ RES
✔ NORES
CICS
NORES Linked

To determine which programs you need to link-edit with Language Environment, you need to know:

1. If the program was compiled with the RES or NORES compiler option
2. If the program uses a reusable environment (established by ILBOSTP0)

For a summary table of load modules with OS/VS COBOL programs that require link-edit, see Figure 46 on page 174.

For additional information on relink-editing a COBOL load module with Language Environment, see Appendix K, “Link-edit example” on page 245.

Applications with COBOL programs compiled RES: If you CALL ILBOSTP0 to establish a reusable environment (that is, you establish an assembler language program as the main program by link-editing it with, and calling, ILBOSTP0), you must link-edit the assembler program with Language Environment.

Applications compiled NORES: Existing OS/VS COBOL programs compiled NORES run without change and provide the same results as before. You do not need to link-edit these programs with Language Environment; however, you will not be able to get IBM service support for these NORES applications unless you link-edit these programs with Language Environment.

Note: If your program contains a CALL identifier statement, the compiler overrides any NORES specification to be RES. This can result in an application with mixed RES and NORES, even if NORES is specified as the installation default.

If you do link-edit programs compiled NORES with Language Environment, see Chapter 8, “Link-editing applications with Language Environment” on page 99 for details on possible changes in behavior.

Applications with COBOL programs compiled RES and NORES: Although this combination was never supported under OS/VS COBOL, in some cases it did work. You **must** link-edit these programs with Language Environment and include at least **one** of the following in the first executed load module that has COBOL:

- A VS COBOL II program

- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- The IGZCBSN run-time library routine
- The IGZCBSO run-time library routine

Determining which programs require upgrade

This section applies to:

✔ RES
✔ NORES
✔ CICS
NORES Linked

On CICS: OS/VS COBOL programs that issue dynamic CALLs when running on CICS will abend with abend code U3504. You can either upgrade all the COBOL programs in the run unit to IBM COBOL or recode the programs without dynamic CALLs.

On non-CICS

You must upgrade OS/VS COBOL programs to IBM COBOL if:

1. The program uses ILC with PL/I
2. The program uses ILC with FORTRAN
3. OS/VS COBOL programs are contained in more than one enclave (or run unit)

You might also want to upgrade the following applications to IBM COBOL:

- Applications that you want to use Language Environment or IBM COBOL features (such as Language Environment callable services or intrinsic functions).
- Applications requiring recoding to run correctly under Language Environment.

ILC with PL/I: OS/VS COBOL programs that CALL or are called by PL/I will not run under Language Environment. They must be upgraded to IBM COBOL.

ILC with FORTRAN: OS/VS COBOL programs that CALL or are called by FORTRAN will not run under Language Environment. They must be upgraded to IBM COBOL.

OS/VS COBOL programs in more than one enclave: Under Language Environment, OS/VS COBOL programs cannot be in more than one enclave. Multiple enclaves can be created in several ways:

- When an OS/VS COBOL program CALLs an assembler program, which in turn issues an SVC LINK to another OS/VS COBOL program.
- When running Language Environment on CMS and using CMSCALL.
- When running OS/VS COBOL applications comprised of programs compiled RES under Language Environment that use ISPF services such as CALL 'ISPLINK' or ISPF SELECT.

For multienclave applications, determine which enclaves contain OS/VS COBOL programs. Only one enclave can contain OS/VS COBOL programs. You must upgrade any OS/VS COBOL programs contained in all other enclaves to IBM COBOL.

Comparing run-time options and specification methods

This section applies to:

✔ RES
✔ NORES
CICS
✔ NORES Linked

This section lists the mechanisms available to specify Language Environment run-time options and gives a brief description of each method.

Specifying Language Environment run-time options

Language Environment provides various methods for specifying run-time options. Three mechanisms are available to specify options:

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

To determine which of the Language Environment 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 Language Environment. Figure 24 lists the specification methods available to specify and override run-time options under Language Environment.

Figure 24. Methods available for specifying run-time options for OS/VS COBOL programs

Main program	CEEDOPT and CEEROPT	CEEUOPT	Invocation	Default assembler user exit
Not link-edited with Language Environment:				
OS/VS COBOL RES	X		X	X
OS/VS COBOL NORES			X	
Link-edited with Language Environment:				
OS/VS COBOL RES	X		X	X
OS/VS COBOL NORES			X	
OS/VS COBOL NORES linked[1]	X		X	X

Note:

1 In this case, the NORES programs exhibit “RES behavior” after link-edit. For more information about the impact of link-editing with Language Environment, see Chapter 8, “Link-editing applications with Language Environment” on page 99.

Installation-wide defaults

The CEEDOPT (non-CICS) and CEECOPT (CICS) assembler files contain default values for all Language Environment 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 (except OS/VS COBOL programs under CICS).

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.

For information on how to set installation-wide default run-time options for Language Environment, see:

- For MVS: *Language Environment Installation and Customization*
- For OS/390: *Language Environment for OS/390 Customization*
- For VM: the *Program Directory*

Region-wide defaults

CEEROPT can be used in the same manner as CEEDOPT and CEECOPT to specify run-time options. CEEROPT can be used to set run-time option defaults for Language Environment in a CICS region, an IMS region, or an MVS batch job region using Library Routine Retention. The fact that CEEROPT resides in its own load module avoids the maintenance problems associated with linking it into a load module containing executable code. CEEROPT will be loaded and merged with the installation run-time options default during region initialization.

CEEROPT is optional. During environment initialization, an attempt to locate CEEROPT is performed. If it is found, the run-time options specified within it will be merged with the installation defaults specified in CEEDOPT or CEECOPT.

For information on how to set region-wide default run-time options for Language Environment, see *Language Environment for OS/390 Customization*.

CEEUOPT application-specific defaults

CEEUOPT is not available to load modules that have an OS/VS COBOL program as the main program. This is true regardless of whether you have link-edited the OS/VS COBOL program with Language Environment or not.

Invocation procedures

For OS/VS COBOL programs that are called directly from the operating system, you can specify run-time options at program invocation using the appropriate operating system mechanism.

On OS/390 and MVS

You can specify options using the PARM option of the JCL EXEC statement.

On CMS

You can specify options using the CMS START command. You can also specify options on the OSRUN command and, when starting an application, after using the GENMOD command.

Note: On CICS, you cannot use the PARM parameters to specify run-time options.

For specific details, see the *Language Environment Programming Guide*.

Order of precedence

It is possible to use all of the above mechanisms for specifying run-time options for a single application. Language Environment enforces the following rules of precedence:

1. Installation-wide defaults specified as nonoverridable at installation time
2. Options specified through the assembler user exit
3. Options specified on invocation
4. Application-specific options (link-edited with the application)
5. Region-wide defaults defined using CEEROPT
6. Installation-wide defaults defined at installation time

Comparing OS/VS COBOL and Language Environment run-time options

The following table describes the support Language Environment provides for OS/VS COBOL run-time options.

Figure 25. Comparison of OS/VS COBOL and Language Environment run-time options

OS/VS COBOL option	Comment
AIXBLD	AIXBLD is supported in the same manner and specified with the same syntax under Language Environment as in OS/VS COBOL.
DEBUG	DEBUG is supported in the same manner under Language Environment as in OS/VS COBOL.
FLOW FLOW=n	FLOW is supported in the same manner under Language Environment as in OS/VS COBOL. FLOW cannot be specified using CEEDOPT or CEEUOPT. It can only be specified at invocation.
QUEUE	QUEUE is not supported.
UPSI	UPSI is supported in the same manner and specified with the same syntax under Language Environment as in OS/VS COBOL.

Closing files in non-COBOL and OS/VS COBOL programs

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

For OS/390 or MVS, if you have OS/VS COBOL programs or assembler programs that do not close files, you can get a C03 abend. In batch, when a COBOL main program is invoked from an assembler program or if COBOL is not the main program, you need to make one of the following changes to avoid getting the C03 abend:

- Add code to close the file.
- Use either a VS COBOL II **or** an IBM COBOL program to open the file.
- If the COBOL program is being called by an assembler program, and if the assembler program is invoked by the MVS or OS/390 batch initiator, front end the application with a COBOL stub so that Language Environment will not free dynamically called load modules at termination (the assembler program and the I/O control blocks it contains stay in storage).
- If the assembler program is opening the file, change it to perform a GETMAIN for the data management control blocks (instead of having the control blocks be part of the load module).

Other environments: For programs run in other environments, such as: TSO, IMS, or preinitialized environments (PIPI or IGZERRE programs), you must explicitly close all files from OS/VS COBOL (or upgrade the OS/VS COBOL programs to IBM COBOL) prior to enclave termination. Language Environment will not automatically close files left open by OS/VS COBOL programs.

Assembler programs that are called by COBOL must close files prior to enclave termination when:

- The assembler program is in a load module that is loaded by a COBOL dynamic CALL.
- The assembler program allocates the storage for the file control block in the assembler program or in a COBOL WORKING-STORAGE data item.

Running in a reusable run-time environment

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Language Environment continues to provide compatibility support for the reusable run-time environment for applications provided by ILBOSTP0.

When using the reusable environment with existing applications under Language Environment, the following restrictions apply:

- The application must be a single enclave application (that is, no SVC LINKs are allowed to other COBOL programs) unless you modify the behavior using the IGZERREO CSECT.

  Note: Modifying the behavior using the IGZERREO CSECT does not remove the restriction that OS/VS COBOL cannot be in more than one enclave.

- Once the reusable environment is established, the assembler driver can call only a COBOL program or non-.Language Environment-conforming assembler program. COBOL can then call any supported language as follows:
  - A COBOL program that is called by the assembler driver cannot use a static call to call another language. A dynamic call to another language is supported.
  - Any COBOL program that is dynamically called by another COBOL program can use a static call or a dynamic call to another language.
- The first high-level language program must be a COBOL program (assembler is not considered a high-level language).
- If an assembler driver is statically linked with, and calls, ILBOSTP0 to provide the reusable environment, it must be link-edited with Language Environment.

Using STOP RUN will end the reusable environment regardless of the method used to initialize it. If ILBOSTP0 is called by an assembler program, control is returned to the caller of the assembler program.

You can improve the performance when using the COBOL reusable environment or you can allow nested enclaves when running in a reusable environment by modifying the IGZERREO CSECT. For additional information, see the:

- For MVS: *Language Environment Installation and Customization*
- For OS/390: *Language Environment for OS/390 Customization*
- For VM: *Program Directory*

Using ILBOSTP0: ILBOSTP0 is still supported so that existing applications that use non-COBOL programs as main programs will continue to run and provide the same results.

Note: Since ILBOSTP0 is AMODE(24), when used under Language Environment, Language Environment will automatically set ALL31(OFF) and STACK(,,BELOW).

Managing dump services

This section applies to:

✔ RES
NORES
✔ CICS
✔ NORES Linked

Dump services are managed by Language Environment. This section describes the support for symbolic dumps, system dumps, and transaction dumps. It also includes information on the Language Environment formatted dump.

OS/VS COBOL symbolic dumps

For those OS/VS COBOL programs that are compiled with the NORES compiler option, dump output generated using SYMDMP remains the same and continues to be directed to SYSDBOUT. If the RES compiler option is used, a CEEDUMP will be produced instead.

System storage dumps and CICS transaction dumps

The PSW and registers information contained in the dump is different when running under Language Environment than when running under the OS/VS COBOL run time.

With OS/VS COBOL, the dump is produced when the error occurs. The PSW and registers content is based on the information available when the error occurs. You could then use the PSW and register information in the dump for problem determination.

With Language Environment, the dump is produced after Language Environment condition management has processed the error. The PSW and registers content is based on the information available when Language Environment completes processing the error, **not** on the information available when the error occurs. To determine the information that was in the PSW and registers at the time of the error, you can either look at the Language Environment formatted dump output or locate and use the Language Environment Condition Information Block and the Language Environment Machine State Information Block in the dump. For details, see the *Language Environment Debugging Guide and Run-Time Messages*.

To obtain a system storage dump or CICS transaction dump under Language Environment, see “Obtain a system dump or a CICS transaction dump” on page 53.

Under both OS/VS COBOL and Language Environment, system storage dump output is directed to either SYSUDUMP, SYSABEND, or SYSMDUMP. CICS transaction dump output is directed to either DFHDUMPA or DFHDUMPB.

Language Environment formatted dumps

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

The Language Environment formatted dump can contain various information, including information on:

- Symbolic dump of variables, arguments, and registers
- File control blocks
- File buffers
- Run-time control blocks
- Storage used by the program

For examples of Language Environment dump output, see *Language Environment Debugging Guide and Run-Time Messages*.

Dump destination on non-CICS

With Language Environment, you can indicate the destination for dump output by providing the CEEDUMP ddname in your JCL or FILEDEF. For the attributes of the CEEDUMP file, see the *Language Environment Programming Guide*.

If the CEEDUMP file is needed and is not defined, Language Environment will dynamically allocate one for you, with the default being:

- On OS/390 and MVS: SYSOUT=*
- On CMS: FILEDEF CEEDUMP PRINTER

- On TSO: ALLOC DD(CEEDUMP) SYSOUT

Dump destination on CICS

All run-time output from Language Environment 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.

Using ILBOABN0 to force an abend

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Under OS/VS COBOL, you can use a CALL to ILBOABN0 to force an immediate abend and obtain a system dump. Under Language Environment, a CALL to ILBOABN0 will continue to force an immediate abend. To produce a system dump, see “Obtain a system dump or a CICS transaction dump” on page 53.

If using the Language Environment version of ILBOABN0, the save area of the program issuing ILBOABN0 is located two levels back from the save area where the actual abend was issued.

OS/VS COBOL programs that use a CALL to ILBOABN0 can continue to CALL ILBOABN0 when compiled with IBM COBOL. However, it is recommended that you use the Language Environment CEE3ABD callable service instead.

Using SORT/MERGE in OS/VS COBOL programs

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Implementation of Language Environment caused changes in the run-time routines that manage OS/VS COBOL SORT/MERGE. If an OS/VS COBOL program initialized the SORT/MERGE, the following changes apply:

- Language Environment will not produce a Language Environment dump if a program check or abnormal termination occurs while in a SORT/MERGE user exit.
- OS/VS COBOL debug data will not be produced if a program check or abnormal termination occurs while in a SORT/MERGE user exit.
- Language Environment will not perform environment cleanup 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 program check occurs while in a SORT/MERGE user exit, the application will end with abend U4036. For information on how to find the state of the registers and PSW at the time of the program check, see *Language Environment Debugging Guide and Run-Time Messages*.

Understanding SYSOUT output changes

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Under Language Environment, SYSOUT output is different than with OS/VS COBOL. Differences are with:

- SYSOUT output sent to a file with RECFM=FB or to any other destination when DCB=(RECFM=FB) is specified on the DD
- OS/VS COBOL trace output sequence
- Maximum message length for DISPLAY UPON SYSOUT

SYSOUT output with RECFM=FB

Under Language Environment, for DISPLAY/TRACE/EXHIBIT SYSOUT output that is sent to a file with RECFM as FB, or to any other destination when DCB=(RECFM=FB) is specified on the DD, the first character is not included in the output, as shown below:

Record format	With Language Environment	With OS/VS COBOL
RECFM=FBA	$01234	$01234
RECFM=FB	01234	$01234

The '$' represents the control character at column 1.

OS/VS COBOL trace output sequence

The trace output sequence has changed under Language Environment. Under Language Environment, only **one** trace entry per record appears, instead of numerous trace entries per record as in OS/VS COBOL. This is especially helpful with intervening DISPLAY SYSOUT output and error messages.

Figure 26 shows the differences between OS/VS COBOL and Language Environment.

```
OS/VS COBOL:
     Record 1 -->  Section1(2), Paragraph1(2), Section2(2)

Language Environment:
     Record 1 -->  Section1
     Record 2 -->  Section1
     Record 3 -->  Paragraph1
     Record 4 -->  Paragraph1
     Record 5 -->  Section2
     Record 6 -->  Section2
```

Figure 26. TRACE output under Language Environment compared to OS/VS COBOL

Communicating with other languages

This section applies to:

✔ RES
✔ NORES
✔ CICS
✔ NORES Linked

This section gives you a high-level overview of ILC considerations for existing OS/VS COBOL programs. For exact details and for the latest information on ILC under Language Environment, see the *Language Environment Writing Interlanguage Applications*.

Interlanguage communication is defined as programs that call or are called by other high-level languages. Assembler is not considered a high-level language; thus, calls to and from assembler programs are not considered ILC. For details on Language Environment's support for calls involving COBOL programs and Assembler programs, see:

- "Run-time support for assembler/COBOL calls on non-CICS" on page 211
- "Run-time support for assembler/COBOL calls on CICS" on page 212

For CICS, you can continue to use EXEC CICS LINK and EXEC CICS XCTL to communicate with other languages.

Figure 27 shows the action required for applications with existing ILC:

Figure 27. Action required for existing applications using ILC

High-level language	Comments
PL/I	OS/VS COBOL programs that CALL or are called by PL/I must be upgraded to IBM COBOL.
FORTRAN	OS/VS COBOL programs that CALL or are called by FORTRAN must be upgraded to IBM COBOL.
C	ILC between OS/VS COBOL programs and C programs is not, and has never been, supported.

Additional CICS considerations

Language Environment provides a set of compatibility library routines that permit you to run OS/VS COBOL applications on CICS in compatibility mode. When you run an OS/VS COBOL application on CICS, the environment that is established for a run unit by the compatibility library routines supports only OS/VS COBOL. This compatibility library does not contain many of the services normally offered under Language Environment. For example, Language Environment run-time options and callable services are not enabled or supported.

Chapter 7. Moving from the VS COBOL II run time

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

- Determining which programs require link-edit
- Determining which programs require upgrade
- Comparing run-time options and specification methods
- Closing files in non-COBOL and OS/VS COBOL programs
- Running in a reusable run-time environment
- Managing messages, abend codes, and dump services
- Using ILBOABN0 to force an abend
- Using SORT/MERGE
- Understanding SYSOUT output changes
- Communicating with other languages
- Initializing the run-time environment
- Determining storage tuning changes
- Additional CICS considerations

Additional information on moving your run time to Language Environment is included in:

- Appendix D, “Applications with COBOL and assembler” on page 209
- Appendix M, “IMS considerations” on page 282
- Appendix N, “CMS considerations” on page 287

Each section in this chapter indicates whether it is applicable to existing VS COBOL II or OS/VS COBOL programs compiled with the RES compiler option, NORES compiler option (including whether the programs are link-edited with Language Environment), and for applications running on CICS, by using the following notation:

This section applies to:

✔ RES
✔ NORES
✔ CICS
✔ NORES Linked

RES An application comprised of programs compiled with RES.

NORES An application comprised of programs compiled with NORES. The NORES programs have not been link-edited with Language Environment.

OR

An application comprised of OS/VS COBOL programs compiled NORES that **is** link-edited with Language Environment, but **does not** contain any of the following:

- A VS COBOL II program
- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- The IGZCBSN or IGZCBSO run-time routine
- A program using the IGZBRDGE routine

CICS An application that runs on CICS.

NORES linked

An application comprised of programs compiled with NORES. The NORES programs have been link-edited with Language Environment and now behave as if they were RES. The application **does** contain at least one of the following:

- A VS COBOL II program
- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- The IGZCBSN or IGZCBSO run-time routine
- A program using the IGZBRDGE routine

Note: For multiple load module applications, if the **first** load module contains one of the above, the application will behave as if it were RES.

Determining which programs require link-edit

This section applies to:

✔ RES
✔ NORES
CICS
NORES Linked

To determine which programs you need to link-edit with Language Environment, you need to know:

1. If the program was compiled with the RES or NORES compiler option
2. If the program specifies the MIXRES run-time option
3. If the program uses a reusable environment (established by ILBOSTP0)
4. If the program uses ILC
5. If you statically call IGZCA2D or IGZCD2A

For a summary table of load modules with VS COBOL II programs that require link-edit, see Figure 47 on page 175.

For additional information on relink-editing a COBOL load module with Language Environment, see Appendix K, “Link-edit example” on page 245.

Applications with COBOL programs compiled RES: If you CALL ILBOSTP0 to establish a reusable environment (that is, you establish an assembler language program as the main program by link-editing it with, and calling, ILBOSTP0), you must link-edit the assembler program with Language Environment or you can change your assembler program to use IGZERRE.

Applications with COBOL programs compiled NORES: Any load modules containing programs compiled with NORES that use the VS COBOL II MIXRES or RTEREUS run-time option must be link-edited with Language Environment. After link-editing the load module with Language Environment, it will have access to all Language Environment services (except callable services) in all but two cases:

1. **OS/VS COBOL NORES without IGZBRDGE:**

   If the program was not link-edited with an object module produced using the IGZBRDGE macro, and if this is the only load module in an application or the first load module in an application with multiple load modules, after link-editing with Language Environment, it will exhibit **NORES** behavior. (That is, the load module will not have access to Language Environment services.) The reason you must link-edit this application with Language Environment is to ensure that the OS/VS COBOL initiated termination, like STOP RUN, will work.

If you need this type of program to have access to Language Environment services, you can either include an IBM COBOL program in the application, or explicitly INCLUDE either the Language Environment IGZCBSN or IGZCBSO library routine when link-editing the application with Language Environment.

2. **OS/VS COBOL RES and OS/VS COBOL NORES without IGZBRDGE:**

   If this is the only load module in an application or the first load module in an application with multiple load modules, you **must** explicitly INCLUDE either the Language Environment IGZCBSN or IGZCBSO library routine when link-editing the load module with Language Environment. Otherwise, the load module is not supported.

   Additionally, any load modules containing VS COBOL II programs that are run in a multitasking environment must be link edited with Language Environment.

Programs that use ILC: Existing VS COBOL II load modules that use ILC with PL/I, C, or FORTRAN require link-edit. For details, see:

- "COBOL and PL/I" on page 91
- "COBOL and C/370" on page 91
- "COBOL and FORTRAN" on page 90

Statically calling IGZCA2D or IGZCD2A: If you call either IGZCA2D or IGZCD2A statically, you must relink your applications with REPLACE statements for these modules.

Determining which programs require upgrade

This section applies to:

✔ RES
✔ NORES
✔ CICS
NORES Linked

CICS: On CICS, if you have any VS COBOL II programs that use the COBOL CALL statement to call OS/VS COBOL programs, you must compile the OS/VS COBOL program with either VS COBOL II or IBM COBOL.

Note: CALLs between OS/VS COBOL and VS COBOL II Release 4 programs on CICS are diagnosed by the VS COBOL II run time. The diagnosis is also provided for VS COBOL II Release 3.2 and Release 3.1 if you have APAR PN18560 applied. If you are running with these levels, you do not have VS COBOL II programs using the COBOL CALL statement to call OS/VS COBOL on CICS.

Non-CICS

On non-CICS, you are not required to upgrade existing VS COBOL II programs to IBM COBOL to run them under Language Environment.

In the following cases, you must upgrade OS/VS COBOL programs to IBM COBOL:

1. If OS/VS COBOL programs use ILC with PL/I
2. If OS/VS COBOL programs use ILC with FORTRAN
3. If OS/VS COBOL programs are contained in more than one enclave (or run unit)

You might also want to upgrade your OS/VS COBOL or VS COBOL II programs to IBM COBOL if you want them to use IBM COBOL or Language Environment features (such as Language Environment callable services, intrinsic functions, or running non-CICS COBOL programs under more than one task in the same address space under MVS or OS/390).

ILC with PL/I: OS/VS COBOL programs that CALL or are called by PL/I will not run under Language Environment. They must be upgraded to IBM COBOL.

ILC with FORTRAN: OS/VS COBOL RES programs that CALL or are called by FORTRAN will not run under Language Environment. They must be upgraded to IBM COBOL and run under Language Environment.

OS/VS COBOL programs in more than one enclave: Under Language Environment, OS/VS COBOL programs cannot be in more than one enclave. Multiple enclaves can be created in several ways:

- When an OS/VS COBOL program CALLs an assembler program, which in turn issues an SVC LINK to another program.
- When running Language Environment on CMS and using CMSCALL
- When running OS/VS COBOL applications comprised of programs compiled RES under Language Environment that use ISPF services such as CALL 'ISPLINK' or ISPF SELECT.

For multienclave applications, determine which enclaves contain OS/VS COBOL programs. Only one enclave can contain OS/VS COBOL programs. You must upgrade any OS/VS COBOL programs contained in all other enclaves to IBM COBOL.

Comparing run-time options and specification methods

This section applies to:

✔ RES
✔ NORES
✔ CICS
✔ NORES Linked

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

Specifying Language Environment run-time options

Language Environment provides various methods for specifying run-time options. Three mechanisms are available to specify options:

- Run-time option CSECTs:
  - CEEDOPT for installation-wide defaults
  - CEEROPT for region-wide defaults
  - 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 load modules comprised of VS COBOL II programs compiled with RES. IGZEOPT is ignored under CICS.

To determine which of the Language Environment 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 Language Environment.

Figure 28 lists the specification methods available to specify and override run-time options under Language Environment.

Figure 28. Methods available for specifying run-time options

Main program	CEEDOPT and CEEROPT	CEEUOPT	Invocation	Default assembler user exit	IGZEOPT
Not link-edited with Language Environment:					
OS/VS COBOL RES	X		X	X	
OS/VS COBOL NORES			X		
VS COBOL II RES	X		X	X	X[1]
VS COBOL II NORES			X		
VS COBOL II NORES			X		X[3]
Link-edited with Language Environment:					
OS/VS COBOL RES	X		X	X	
OS/VS COBOL NORES			X		
OS/VS COBOL NORES linked[4]	X		X	X	
VS COBOL II NORES	X	X	X	X	
VS COBOL II RES	X	X	X	X	X[1,2]

Note:

1 IGZEOPT is ignored when running on CICS.

2 See Figure 29 on page 76 for details on when the run-time options specified using IGZEOPT are in effect.

3 VS COBOL II NORES programs that do not specify the MIXRES run-time option and that are not link-edited with Language Environment can continue to use IGZEOPT.

4 In this case, the NORES programs will exhibit "RES behavior" after link-edit. For more information on the implication of link-editing with Language Environment, see Chapter 8, "Link-editing applications with Language Environment" on page 99.

Installation-wide defaults

The CEEDOPT (non-CICS) and CEECOPT (CICS) assembler files contain default values for all Language Environment 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.

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.

For information on how to set installation-wide default run-time options see the:

- For MVS: *Language Environment Installation and Customization*
- For OS/390: *Language Environment for OS/390 Customization*
- For VM: *Program Directory*

Region-wide defaults

CEEROPT can be used in the same manner as CEEDOPT and CEECOPT to specify run-time options. CEEROPT can be used to set run-time option defaults for Language Environment in a CICS region, an IMS region, or an MVS batch job region using Library Routine Retention. The fact that CEEROPT resides in its own load module avoids the maintenance problems associated with linking it into a load

module containing executable code. CEEROPT will be loaded and merged with the installation run-time options default during region initialization.

CEEROPT is optional. During environment initialization, an attempt to locate CEEROPT is performed. If it is found, the run-time options specified within it will be merged with the installation defaults specified in CEEDOPT or CEECOPT.

For information on how to set region-wide default run-time options for Language Environment, see *Language Environment for OS/390 Customization*.

Application-specific defaults

Language Environment provides an assembler file, CEEUOPT, 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.

CEEUOPT is not available to applications that have an OS/VS COBOL program as the main program. This is true regardless of whether you have link-edited the OS/VS COBOL program with Language Environment or not.

Invocation procedures

Run-time options can also be specified at program invocation using the appropriate operating system mechanism. (COBOL programs must be called directly from the operating system to specify run-time options at invocation.)

On OS/390 and MVS
You can specify options using the PARM option of the JCL EXEC statement.

On CMS You can specify options using the CMS START command. You can also specify options on the OSRUN command and, when starting an application, after using the GENMOD command.

Note: On CICS and IMS, you cannot use the PARM parameters to specify run-time options.

For specific details, see the *Language Environment Programming Guide*.

Order of precedence

It is possible to use all of the above mechanisms for specifying run-time options for a single application. Language Environment enforces the following rules of precedence:

1. Installation-wide defaults specified as nonoverridable at installation time
2. Options specified through the assembler user exit
3. Options specified on invocation
4. Application-specific options (link-edited with the application)
5. Region-wide defaults defined using CEEROPT
6. Installation-wide defaults defined at installation time

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 upgrade to IBM COBOL or link-edit with Language Environment. As programs in the application are modified, it is recommended that the appropriate Language Environment options module be included in the application.

Applications with COBOL programs compiled RES (non-CICS): The VS COBOL II application-specific run-time options module, IGZEOPT, is available to RES applications, running under Language Environment. After link-editing an application with Language Environment, it is possible for both IGZEOPT and CEEUOPT to exist in the same application. Figure 29 shows the relationship between IGZEOPT and CEEUOPT for applications that are link-edited with Language Environment.

Figure 29. Application-specific options for existing applications compiled with VS COBOL II and link-edited with Language Environment

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 Language Environment run-time option.
No	Yes	The run-time options specified in CEEUOPT are in affect.
Yes	Yes	The run-time options specified in CEEUOPT are in affect. IGZEOPT is ignored. No error message is issued.

It is recommended that you replace IGZEOPT with CEEUOPT when you link-edit these applications with Language Environment.

Applications with COBOL programs 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. IGZEOPT is also available to VS COBOL II NORES applications, if the MIXRES run-time option is not used. Once these applications are link-edited with Language Environment, IGZEOPD and IGZEOPT are ignored.

Applications running on CICS: When running under Language Environment, IGZEOPT is ignored on CICS. Language Environment issues a warning message if IGZEOPT is specified in applications running on CICS.

Comparing VS COBOL II and Language Environment options

The following table describes how VS COBOL II run-time options are either supported or changed when running with Language Environment. The differences in behavior (if any) are described in the comments column.

Figure 30 (Page 1 of 2). Comparison of VS COBOL II and Language Environment run-time options

VS COBOL II option	Language Environment option	Comment
AIXBLD	AIXBLD	Invokes the access method services (AMSs) for VSAM indexed and relative data sets to complete the file and index definition procedures for COBOL routines. With Language Environment, you do not need to specify ddname SYSPRINT when running on OS/390 or MVS. The Access Method Services (AMS) messages are directed to the ddname specified in the Language Environment MSGFILE run-time option (default is SYSOUT). Under CMS, the messages are erased (which is the same behavior as VS COBOL II). AIXBLD is not applicable on CICS.
DEBUG	DEBUG	This option is specified with the same syntax and supported in the same manner under Language Environment as in VS COBOL II.
LANGUAGE	NATLANG	NATLANG replaces LANGUAGE, which is a VS COBOL II installation option. Language Environment 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. The Language Environment Library Routine Retention facility can provide similar function as LIBKEEP. For more information, see “Existing applications using LIBKEEP” on page 92. The LIBKEEP option is not applicable on CICS.
MIXRES		MIXRES is not supported under Language Environment. The MIXRES option is not applicable on CICS.
RTEREUS	RTEREUS	RTEREUS is supported in the same manner under Language Environment as in VS COBOL II. RTEREUS is not recommended as an installation default under Language Environment. For important considerations before using RTEREUS, see “Recommended run-time options for non-CICS applications” on page 46, and “Precautions if establishing a reusable environment under IMS” on page 79. The RTEREUS option is not applicable on CICS.
SIMVRD	SIMVRD	This option is specified with the same syntax and supported in the same manner in Language Environment as in VS COBOL II. The SIMVRD option is not applicable on CICS.
SPOUT	RPTOPTS(ON) RPTSTG(ON)	SPOUT is mapped to the Language Environment options RPTOPTS and RPTSTG. Both storage reports (RPTSTG) and options report (RPTOPTS) are directed to the ddname specified in MSGFILE (default SYSOUT). The report formats are different under Language Environment than with VS COBOL II. For more information, see “Determining storage tuning changes” on page 93.
SSRANGE	CHECK(ON)	SSRANGE is mapped directly to CHECK.

Figure 30 (Page 2 of 2). Comparison of VS COBOL II and Language Environment run-time options

VS COBOL II option	Language Environment option	Comment
STAE	TRAP(ON)	STAE is mapped directly to TRAP(ON). For non-CICS applications, TRAP(ON) causes both ESTAE and ESPIE to be issued. Under VS COBOL II, STAE only causes ESTAE to be issued. Note, that under VS COBOL II STAE is optional. Under Language Environment Release 1.9 (OS/390 V2R6) or later, in non-CICS, you can specify TRAP(ON,NOSPIE) in which case only an ESTAE will be issued.
UPSI	UPSI	This option is specified with the same syntax and supported in the same manner in Language Environment 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.

Closing files in non-COBOL and OS/VS COBOL programs

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

For MVS or OS/390, if you have OS/VS COBOL programs or assembler programs that do not close files, you can get a C03 abend. In batch, when a COBOL main program is invoked from an assembler program or if COBOL is not the main program, you need to make one of the following changes to avoid getting the C03 abend:

- Add code to close the file.
- Use either a VS COBOL II or an IBM COBOL program to open the file.
- If the COBOL program is being called by an assembler program, and if the assembler program is invoked by the MVS or OS/390 batch initiator, front end the application with a COBOL stub so that Language Environment will not free dynamically called load modules at termination (the assembler program and the I/O control blocks it contains stay in storage).
- If the assembler program is opening the file, change it to perform a GETMAIN for the data management control blocks (instead of having the control blocks be part of the load module).

Other environments: For programs run in other environments, such as: TSO, IMS, or preinitialized environments (PIPI or IGZERRE programs), you must explicitly close all files from OS/VS COBOL (or upgrade the OS/VS COBOL programs to IBM COBOL) prior to enclave termination. Language Environment will not automatically close files left open by COBOL programs when running in a reusable environment.

Assembler programs that are called by COBOL must close files prior to enclave termination when:

- The assembler program is in a load module that is loaded by a COBOL dynamic CALL.
- The assembler program allocates the storage for the file control block in the assembler program or in a COBOL WORKING-STORAGE data item.

Running in a reusable run-time environment

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Language Environment 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:

- Call IGZERRE from an assembler program
- Call ILBOSTP0 from an assembler program
- Specify the RTEREUS run-time option

When using the reusable environment with existing applications under Language Environment, the following restrictions apply:

- The application must be a single enclave application unless you modify the behavior using the IGZERREO CSECT.

  Note: Modifying the behavior using the IGZERREO CSECT does not remove the restriction that OS/VS COBOL cannot be in more than one enclave.

- Once the reusable environment is established, the assembler driver can call only a COBOL program or non-Language Environment conforming assembler program. COBOL can then call any supported language.
- The first high-level language program must be a COBOL program (assembler is not considered a high-level language).
- If an assembler driver is statically linked with, and calls, IGZERRE or ILBOSTP0 to provide the reusable environment, it must be link-edited with Language Environment.
- If a COBOL NORES program is running with the RTEREUS run-time option in effect, it must be link-edited with Language Environment.

Using STOP RUN will end the reusable environment regardless of the method used to initialize it. If IGZERRE or ILBOSTP0 is called by an assembler program, control is returned to the caller of the assembler program. If you use the Language Environment RTEREUS run-time option, control is returned to the caller of the caller of the first COBOL program.

You can improve the performance when using the COBOL reusable environment or you can allow nested enclaves when running in a reusable environment by modifying the IGZERREO CSECT. For additional information, see the:

- For MVS: *Language Environment Installation and Customization*
- For OS/390: *Language Environment for OS/390 Customization*
- For VM: *Language Environment Program Directory*

Precautions if establishing a reusable environment under IMS: The reusable environment is not recommended under IMS. If you do choose to initiate a reusable environment, then:

- Use a reusable environment only if your application consists of a small number of different programs. Otherwise, storage will accumulate, diminish, and eventually bring down the IMS region.
- Preload all programs that receive control from IMS (such as COBOL main or assembler). If not preloaded, results are unpredictable.

For additional details on using RTEREUS, see “Recommended run-time options for non-CICS applications” on page 46.

Using IGZERRE

You can continue to use IGZERRE to explicitly drive the initialization and termination functions of COBOL. However, you need to be aware of changes in three of the return codes.

On return from IGZERRE, register 15 contains a return code. Figure 31 shows the changes when running with Language Environment:

Figure 31. Return code changes for IGZERRE

Return code	Comments
0	No change.
4	Is issued if Language Environment 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 MVS and CMS*.

Using ILBOSTP0

ILBOSTP0 is still 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 ILBOSTP0 to initialize the COBOL reusable environment, Language Environment will automatically set ALL31(OFF) and STACK(,,BELOW), as long as the application-specific routine CEEUOPT is **not** linked with the load module.

If you link CEEUOPT with the load module, 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 Language Environment diagnoses this conflict with message CEE3615I. Running the application with these conflicts will yield unpredictable results.

Using RTEREUS

The RTEREUS run-time option for initializing the COBOL reusable environment is supported by Language Environment, although it is not recommended as an installation default. For important considerations, see the RTEREUS discussion under “Recommended run-time options for non-CICS applications” on page 46. For RES environments, the behavior of RTEREUS is the same as it was in VS COBOL II with the exception of using SVC LINK (for example, ISPF "SELECT PGM()").

Under Language Environment, if an assembler program issues an SVC LINK while running in a reusable environment, Language Environment raises a condition resulting in severity-3 message IGZ0168S. With VS COBOL II, no error is detected, but the secondary run unit's environment is not reusable. You can modify the behavior under Language Environment to be similar to the behavior under VS COBOL II by modifying the IGZERREO CSECT. However, a STOP RUN in the nested enclave will only terminate the nested enclave and not the parent enclave.

If the operating system is the invoker of the first COBOL program, RTEREUS will be ignored.

You can specify RTEREUS using the installation-wide default routine CEEDOPT, the application-specific options routine CEEUOPT, or in the assembler user exit.

For important IMS considerations see “Precautions if establishing a reusable environment under IMS” on page 79.

Managing messages, abend codes, and dump services

This section applies to:

✔ RES
NORES
✔ CICS
✔ NORES Linked

Messages, abend codes, and dump services are managed differently for applications running under Language Environment than when running under the VS COBOL II or OS/VS COBOL libraries.

Run-time messages

Two factors affect run-time messages for applications running with Language Environment:

- If the messages are issued by the OS/VS COBOL compatibility library routines (prefixed IKF)
- Status of the Language Environment run-time initialization process for messages issued by COBOL-specific library routines, including VS COBOL II compatibility routines (prefixed IGZ)

Messages prefixed with IKF

There are no differences for OS/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 Language Environment. The messages appear synchronously and are written using OS write-to-programmer. This is true regardless of the state of the run-time initialization process.

Messages prefixed with IGZ

VS COBOL II messages are managed depending on the status of the Language Environment run-time initialization process.

- On non-CICS, if the run-time initialization process is not complete, messages are routed as directed by OS write-to-programmer.

- On non-CICS, if the run-time initialization process is complete, then messages are directed to the file *ddname* specified on the Language Environment MSGFILE run-time option, which is defaulted to SYSOUT.
  - On OS/390 and MVS: SYSOUT=*
  - On CMS: FILEDEF SYSOUT TERMINAL
  - Under TSO: ALLOC DD(SYSOUT) DA(*)

  If MSGFILE ddname is not defined, it will be dynamically allocated with Language Environment defined file attributes. For details, see *Language Environment Programming Guide*.
- On CICS, all run-time output from Language Environment 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, date and time. Run-time output is no longer written to a temporary storage queue.

  Note: If you would like to have the run-time output directed to a temporary storage queue, you can define the transient data queue CESE as an intrapartition data queue and specify a transaction that reads records from CESE, strips off the header information, and writes the rest of the record to a temporary storage queue.

All COBOL-specific messages will be prefixed by IGZ, followed by a four-digit message number, and the level of severity. For example, VS COBOL II message number IGZ020I is message number IGZ0020W under Language Environment. The VS COBOL II informational message becomes a Language Environment severity-1 warning message. The VS COBOL II severe message, which resulted in a 1xxx user abend, becomes either a Language Environment severity-3 severe condition or severity-4 critical condition.

Severity 1 (W)
A warning message. The Language Environment default condition handling action will format and issue the message and then continue program execution.

Severity 2 (E)
An error message. The Language Environment default condition handling action will percolate the condition, issue a message, and terminate the enclave.

Severity 3 (S)
A severe error message. The Language Environment default condition handling action will percolate the condition, issue a message, and terminate the enclave.

Severity 4 (C)
A critical error message. The Language Environment default condition handling action will percolate the condition, issue a message, and terminate the enclave.

Timing of abend for run-time detected errors

The timing of abends between Language Environment and VS COBOL II is different for run-time detected errors (such as IGZ0061S division by zero or IGZ0006S subscript out of range). This difference in the timing of abends affects the behavior of CICS HANDLE ABEND.

Under Language Environment, the following events occur when there is a run-time detected error (with ABTERMENC(ABEND) in effect):

1. A Language Environment software-generated condition is signalled.
2. The Language Environment condition manager gives control to any user condition handlers that have been registered.
3. If the condition has not been handled, the enclave is terminated and Language Environment issues a 4038 abend.

With VS COBOL II, the following events occur when there is a run-time detected error:

1. An error message is written.
2. An abend 1xxx is issued.

Under Language Environment, when a run-time detected error occurs, the enclave (run unit) that contains the code is terminated before the abend is issued. Thus, code at a label referenced in the CICS HANDLE ABEND command does not get control.

Under VS COBOL II, an abend is issued while the run unit still exists, so code at the HANDLE ABEND label is executed.

For behavior compatible with VS COBOL II, use the sample user condition handler code, CEEWUCHA, that is provided with Language Environment in the SCEESAMP data set.

When using CEEWUCHA under Language Environment, the following events occur when there is a run-time detected error (with ABTERMENC(ABEND) in effect):

- A Language Environment software generated condition is signalled.
- The Language Environment condition manager gives control to any user condition handlers that have been registered.
- The CEEWUCHA user condition handler gets control and causes the following to occur:
  - An error message is written.
  - A Language Environment dump is produced.
  - An abend 1xxx is issued.

Abend codes

To see user 1xxx abends similar to what you received under VS COBOL II (where xxx is the IGZ message number) for unhandled Language Environment software generated conditions, do **one** of the following:

- Modify the assembler user exit (CEEBXITA) by copying the code from the COBOL sample assembler user exit (CEEBX05A) into CEEBXITA.
- Use the sample user condition handler CEEWUCHA as described in “Using CEEWUCHA” on page 84.

Using CEEWUCHA

CEEWUCHA is a sample user condition handler that you can use to alter the default behavior of Language Environment to get behavior that is similar to VS COBOL II.

CEEWUCHA contains code to do the following:

- Provide compatibility with existing VS COBOL II applications running under CICS by allowing EXEC CICS HANDLE ABEND LABEL statements to get control when a run-time detected error occurs (such as IGZ0061S, division by zero).
- Convert all unhandled run-time detected errors to the corresponding user 1xxx abend issued by VS COBOL II.
- Suppress all IGZ0014W messages, which are generated when IGZETUN or IGZEOPT is link edited with a VS COBOL II application.

To use CEEWUCHA:

1. Use the CEEWWCHA sample SMP job to assemble and link edit CEEWUCHA.
2. On CICS, define CEEWUCHA in the PPT for your CICS region.
3. Specify USRHDLR(CEEWUCHA) in either:
   - On CICS, CEECOPT (to apply to the entire CICS region)
   - On non-CICS, CEEDOPT (to apply to all applications)
   - On CICS or non-CICS, CEEUOPT and link-edit it with the individual applications

Dump services

Dump services are managed by Language Environment. This section describes the changes for symbolic dumps, formatted dumps, and system dumps.

OS/VS COBOL symbolic dumps

For those OS/VS COBOL programs that are compiled with the NORES compiler option, dump output generated using SYMDMP remains the same and continues to be directed to SYSDBOUT. If the RES compiler option is used, a CEEDUMP will be produced instead.

VS COBOL II FDUMPs

For VS COBOL II programs compiled with the FDUMP option, you could get a formatted dump of COBOL WORKING-STORAGE. VS COBOL II FDUMPs were directed to the SYSDBOUT data set. For details on dump destination under Language Environment, see “Language Environment formatted dumps” on page 85.

When a VS COBOL II program that is compiled with OPT and FDUMP and which contains ODO data structures is run under Language Environment/370 and abends, level-1 items will be dumped as group items in the Language Environment/370 formatted dump.

IBM COBOL formatting dumps

The IBM COBOL TEST(SYM) compiler option provides the same function as the VS COBOL II FDUMP compiler option. When you specify the SYM suboption of the TEST compiler option, Language Environment can provide you with output similar to FDUMP output as part of the Language Environment formatted dump.

Language Environment formatted dumps

Dump formats and destinations for dump output are different under Language Environment than with VS COBOL II.

The Language Environment formatted dump can contain various information, including information on:

- Symbolic dump of variables, arguments, and registers
- File control blocks
- File buffers
- Run-time control blocks
- Storage used by the program

For examples of Language Environment dump output, see *Language Environment Debugging Guide and Run-Time Messages*.

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

Dump destination on non-CICS: With Language Environment, you can indicate the destination for dump output by providing the CEEDUMP ddname in your JCL or FILEDEF. For the attributes of the CEEDUMP file, see the *Language Environment Programming Guide*.

If the CEEDUMP file is needed and is not defined, Language Environment will dynamically allocate one for you, with the default being:

- On OS/390 and MVS: SYSOUT=*
- On CMS: FILEDEF CEEDUMP PRINTER
- On TSO: ALLOC DD(CEEDUMP) SYSOUT

Dump destination on CICS: All run-time output from Language Environment 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.

System storage dumps

The PSW and registers information contained in the system storage dump is different when running under Language Environment than when running under the VS COBOL II run time.

With VS COBOL II, the dump is produced when the error occurs. The PSW and registers content is based on the information available when the error occurs. You could then use the PSW and register information in the dump for problem determination.

With Language Environment, the dump is produced after Language Environment condition management has processed the error. The PSW and registers content is

based on the information available when Language Environment completes processing the error, **not** on the information available when the error occurs. To determine the information that was in the PSW and registers at the time of the error, you can either look at the Language Environment formatted dump output or use the Language Environment condition information block and the Language Environment machine state information block. For details, see the *Language Environment Debugging Guide and Run-Time Messages*.

To obtain a system storage dump, see "Obtain a system dump or a CICS transaction dump" on page 53.

On OS/390 and MVS, dump output is directed to SYSUDUMP, SYSMDUMP, or SYSABEND. On other systems, VS COBOL II run-time dump output is directed to SYSABOUT. (Note, SYSABOUT is not used under Language Environment, even for VS COBOL II programs.)

Using ILBOABN0 to force an abend

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Under OS/VS COBOL and VS COBOL II, you can use a CALL to ILBOABN0 to force an immediate abend and obtain a system dump. Under Language Environment, a CALL to ILBOABN0 will continue to force an immediate abend. To produce a system dump, see "Obtain a system dump or a CICS transaction dump" on page 53.

If using the Language Environment version of ILBOABN0, the save area of the program issuing ILBOABN0 is located two levels back from the save area where the actual abend was issued.

Programs converted to IBM COBOL that use a CALL to ILBOABN0 can continue to CALL ILBOABN0. However, it is recommended that you use the Language Environment CEE3ABD callable service instead.

Using SORT/MERGE

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Implementation of Language Environment caused changes in the run-time routines that manage SORT/MERGE for OS/VS COBOL programs and VS COBOL II programs.

In OS/VS COBOL programs

When an OS/VS COBOL program initialized the SORT/MERGE, the following changes apply:

- Language Environment will not produce a Language Environment dump if a program check or abnormal termination occurs while in a SORT/MERGE user exit

- OS/VS COBOL debug data will not be produced if a program check or abnormal termination occurs while in a SORT/MERGE user exit.
- Language Environment will not perform environment cleanup 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 program check occurs while in a SORT/MERGE user exit, the application will end with abend U4036. For information on how to find the state of the registers and PSW at the time of the program check, see *Language Environment Debugging Guide and Run-Time Messages*.

In VS COBOL II subprograms

Using a GOBACK statement in an input or output procedure that is used with a SORT or MERGE statement for VS COBOL II subprograms behaves differently under Language Environment than under the VS COBOL II run time.

When running VS COBOL II programs under Language Environment, the difference in behavior occurs when:

- A COBOL **subprogram** issues a SORT or MERGE statement with an input procedure and/or an output procedure
- While the SORT or MERGE statement is being executed and an input procedure or an output procedure has been invoked, a GOBACK is done in the compile unit that initiated the SORT or MERGE statement

Under Language Environment, severe condition IGZ0012S is issued. Under VS COBOL II, the subprogram will return to its caller without an error.

For example:

```
⋮
    SORT SD01
       ASCENDING KEY A3
       USING INP1
       OUTPUT PROCEDURE OUTPRO1.
⋮
OUTPRO1 SECTION.
⋮
    GOBACK.
    * With Language Environment, this GOBACK statement will
    * cause condition IGZ0012S.
```

Note: With the above scenario, with VS COBOL II (as with Language Environment), if the COBOL program issuing the SORT/MERGE is a **main** program, an error message is generated.

Understanding SYSOUT output changes

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

Under Language Environment, DISPLAY UPON SYSOUT is different than in OS/VS COBOL and VS COBOL II as follows:

- SYSOUT output sent to a file with RECFM=FB or to any other destination when DCB=(RECFM=FB) is specified on the DD
- OS/VS COBOL trace output sequence

DISPLAY UPON SYSOUT and DD definitions

Language Environment handles DISPLAY UPON SYSOUT differently than VS COBOL II:

- Under VS COBOL II, if the DD card is missing, VS COBOL II issues an IGZ message and abends. Under Language Environment, Language Environment allocates the DD and the DISPLAY is successful.
- Under VS COBOL II, if the value of the OUTDD is different in VS COBOL II programs using DISPLAY, the DD from the first program encountered with a DISPLAY is the DD used for the life of the application. Under Language Environment, Language Environment will allocate a DD for each unique OUTDD you specify.

SYSOUT output with RECFM=FB

Under Language Environment, for TRACE/EXHIBIT output (for OS/VS COBOL programs) and DISPLAY UPON SYSOUT output (for both OS/VS COBOL programs and VS COBOL II programs) sent to a file with RECFM as FB, or to any other destination when DCB=(RECFM=FB) is specified on the DD, the first character is not included in the output, as shown below:

Record format	VS COBOL II	Language Environment
RECFM=FBA	$01234	$01234
RECFM=FB	$01234	01234

The '$' represents the control character at column 1. The default attributes of DISPLAY are unchanged.

We recommend that you use RECFM=FBA for SYSOUT directed to a file.

OS/VS COBOL Trace output sequence

The trace output sequence has changed under Language Environment. In Language Environment, only **one** trace entry per record appears, instead of numerous trace entries per record as in OS/VS COBOL programs running under VS COBOL II. This is especially helpful with intervening DISPLAY SYSOUT output and error messages.

Figure 32 on page 89 shows the difference between the OS/VS COBOL and Language Environment formats.

```
OS/VS COBOL Running under VS COBOL II:

     Record 1 -->  Section1(2), Paragraph1(2), Section2(2)

Language Environment:

     Record 1 -->  Section1
     Record 2 -->  Section1
     Record 3 -->  Paragraph1
     Record 4 -->  Paragraph1
     Record 5 -->  Section2
     Record 6 -->  Section2
```

Figure 32. TRACE output under Language Environment compared to OS/VS COBOL

Communicating with other languages

This section applies to:

✔ RES
✔ NORES
✔ CICS
✔ NORES Linked

This section gives you a high-level overview of ILC considerations for existing OS/VS COBOL programs and VS COBOL II programs running under the VS COBOL II run time. For exact details and for the latest information on ILC under Language Environment, see the *Language Environment Writing Interlanguage Applications*.

Interlanguage communication is defined as programs that call or are called by other high-level languages. Assembler is not considered a high-level language; thus, calls to and from assembler programs are not considered ILC. For details on Language Environment's support for calls involving COBOL programs and Assembler programs, see:

- "Run-time support for assembler/COBOL calls on non-CICS" on page 211
- "Run-time support for assembler/COBOL calls on CICS" on page 212

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

For CICS, you can continue to use EXEC CICS LINK and EXEC CICS XCTL to communicate with other languages.

General ILC considerations

COBOL DISPLAY output (non-CICS)

When you run a COBOL-C ILC application under Language Environment, the SYSOUT DD will remain open until Language Environment terminates. This impacts any ILC application that has the following scenario:

1. A C main program calls a COBOL program.
2. The COBOL program uses the DISPLAY statement. The output from the DISPLAY statement is written to a data set associated with SYSOUT DD. The COBOL program returns to the C program.

3. The C program opens the data set associated with the SYSOUT DD and reads the records written.

The above scenario worked with the VS COBOL II and pre-Language Environment C libraries because the VS COBOL II program was the main program (and thus, the VS COBOL II run time closed the SYSOUT DD when the COBOL program ended). Under Language Environment, the COBOL program is a subprogram, and Language Environment does not close the SYSOUT DD when the COBOL program returns to the C program.

Effect of STOP RUN (non-CICS)

Under Language Environment, when a COBOL program issues a STOP RUN after being called from a C, PL/I, or FORTRAN program, the entire environment terminates. Under VS COBOL II, only the COBOL run time goes down and the calling program (C, PL/I, or FORTRAN) continues to run.

COBOL and FORTRAN

Note: This section is not applicable to CICS. FORTRAN programs running on CICS has never been supported by IBM.

ILC is supported between the following COBOL and FORTRAN releases if link edited with Language Environment:

- VS COBOL II Release 3 and later (static CALLs only)
- COBOL/370
- COBOL for MVS & VM
- COBOL for OS/390 & VM
- VS FORTRAN Version 1 (except for modules compiled with Release 1 or Release 2 and either are subprograms that receive character arguments, or pass character arguments to subprograms).
- VS FORTRAN Version 2 (except modules compiled with Release 5 or Release 6) that:

  Contain parallel language constructs
  Specify the PARALLEL compiler option
  Invoke parallel callable services (PEORIG, PEPOST, PEWAIT, PETERM, PLCOND, PLFREE, PLLOCK, PLORIG, or PLTERM)

- FORTRAN IV G1
- FORTRAN IV H Extended

With Language Environment, OS/VS COBOL RES programs that CALL or are called by FORTRAN are not supported.

Existing NORES applications containing CALLs between COBOL and FORTRAN will continue to run as before, subject to the existing COBOL-FORTRAN ILC rules. However, these applications are not supported if the FORTRAN program is link-edited with Language Environment.

COBOL and PL/I

If you follow the link-edit requirements below, Language Environment will support ILC between the following combinations of COBOL and PL/I:

- VS COBOL II Version 1 Release 3 and later
- COBOL/370
- COBOL for MVS & VM
- COBOL for OS/390 & VM
- OS PL/I Version 1 Release 5.1
- OS PL/I Version 2
- PL/I for MVS & VM

You must link-edit any existing, supported PL/I and COBOL ILC applications either with the PL/I conversion tool (APAR PN46223) while running on OS PL/I Version 2 Release 3 or with Language Environment. (For the PL/I conversion tool to work, you also must install the appropriate USERMOD, COBOL service, and PL/I service. For details, see the *IBM PL/I for MVS & VM Migration Guide*.) You can use either method except in the following cases, when you must link-edit with Language Environment:

- When the application contains COBOL NORES programs.
- When the application requires COBOL program-specific run-time options or space tuning.
- When the application contains PL/I programs that use SORT
- When the application uses the PL/I Shared Library

Note: ILC between OS/VS COBOL and PL/I is not supported. You must upgrade any OS/VS COBOL program containing ILC with PL/I to IBM COBOL.

Difference in behavior for dynamically CALLed RENT programs: For VS COBOL II programs that are:

- Compiled with the RENT option and
- Dynamically CALLed from OS/VS COBOL, VS COBOL II, COBOL/370, or IBM COBOL and then
- Fetched and called by PL/I and

the same copy of WORKING-STORAGE is used for each call. And, the program is entered in its last-used state, unless there is an intervening CANCEL.

For additional information on interlanguage communication between COBOL and PL/I, see *Language Environment Writing Interlanguage Applications* and the *PL/I for MVS & VM Compiler and Run-Time Migration Guide*.

COBOL and C/370

If you follow the link-edit requirements, Language Environment supports ILC between the following combinations of COBOL and C programs:

- VS COBOL II Version 1 Release 3 and later
- COBOL/370
- COBOL for MVS & VM
- COBOL for OS/390 & VM
- C/370 Version 1 (5688-040)
- C/370 Version 2 (5688-187)

- OS/390 C/C++

You must link-edit any existing, supported C and COBOL ILC applications either with:

- Language Environment
- The C conversion tool (APAR PN74931) while running on C/370 Version 2. (For the C conversion tool to work on Language Environment Release 5, you must also install apar PN77483. The function is in the base for Language Environment Release 6 and later.)

You can use either method, except in the following cases, when you must link-edit with Language Environment:

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

Difference in behavior for dynamically CALLed RENT programs: For VS COBOL II programs that are:

- Compiled with the RENT option and
- Dynamically CALLed from OS/VS COBOL, VS COBOL II, COBOL/370, or IBM COBOL and then
- Fetched and called by C and

the same copy of WORKING-STORAGE is used for each call. And, the program is entered in its last-used state, unless there is an intervening CANCEL.

For additional information on interlanguage communication between COBOL and C, see *Language Environment Writing Interlanguage Applications* and the *IBM C/370 Migration Guide*.

Initializing the run-time environment

This section applies to:

✔ RES
NORES
CICS
✔ NORES Linked

The methods available for initializing the run-time environment are different under Language Environment than under VS COBOL II.

Existing applications using LIBKEEP

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 Language Environment run-time environment does not support the LIBKEEP run-time option.

You can obtain similar performance for the main program environment by using the Language Environment Library Routine Retention (LRR) facility. To use LRR in an IMS region, you need to put CEELRRIN in the IMS preinit list. For details on how to do this, see *Language Environment Customization*.

To use LRR within an application program, you call an assembler program that contains the CEELRR macro.

Language Environment provides two sample programs: one to initialize LRR and one to terminate LRR. The sample program names are CEELRRIN and CEELRRTR, and they are in the SCEESAMP library.

For information on LRR, see the *Language Environment Programming Guide* and the customization book for your platform.

Considerations for Language Environment preinitialization

Language Environment preinitialization 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 preinitialized environment using the CEEPIPI(TERM...) service.

To use the Language Environment preinitialization main program services, VS COBOL II programs and OS/VS COBOL programs cannot be the target of CEEPIPI(CALL...). However VS COBOL II programs and OS/VS COBOL programs can be the target of any COBOL CALL statements in a preinitialized environment.

When running an application under the VS COBOL II run time, the data sets used by the functions ACCEPT SYSIN, DISPLAY SYSOUT, and DISPLAY SYSPUNCH are closed after the completion of each invocation of a main program. Thus, the data set content is externally available. For example, in OS/390 and MVS batch environments, closing SYSOUT commonly causes its data to become part of the JOB output. Also, the run-time messages and dumps can be received at the end of each execution.

When running with the Language Environment preinitialization 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 preinitialized environment.

For more information on Language Environment preinitialized services, including information on how to initialize the run-time environment for subprograms, see the *Language Environment Programming Guide*.

Determining storage tuning changes

This section applies to:

✔ RES
NORES
✔ CICS
✔ NORES Linked

The existing methods (the IGZTUNE macro and the SPOUT and WSCLEAR run-time options) for space management and tuning are not available under Language Environment. Language Environment services now control space management and tuning using the RPTOPTS, RPTSTG, STORAGE, ANYHEAP, BELOWHEAP, HEAP, LIBSTACK, and STACK run-time options.

With VS COBOL II, storage tuning is done at the process level; under Language Environment storage tuning is done on an enclave basis. For details of space management and tuning with Language Environment, see the *Language Environment Programming Guide*.

Alternatives to IGZTUNE

If the CSECT produced by assembling the space tuning CSECT IGZETUN is detected in a load module, a warning-level message is issued and the CSECT is not used. Other Language Environment facilities provide storage management capabilities at run-time instead of link-edit time.

Note: For applications using IGZETUN, see “Using CEEWUCHA” on page 84 for details on how to suppress the warning-level message and the overhead of writing it to the Language Environment MSGFILE.

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 for use by Language Environment and COBOL library routines, which can be located anywhere and used for control blocks.

BELOWHEAP Manages heap storage for use by Language Environment and COBOL library routines, which is located below the 16M byte line and used for control blocks and I/O buffers.

STACK Manages stack storage for use by Language Environment, COBOL data items in the LOCAL-STORAGE SECTION, and COBOL library routines, which can be used for dynamic storage area (DSAs).

LIBSTACK Manages stack storage for use by Language Environment and COBOL library routines, which is located below the 16M byte line and used for DSAs.

The RPTSTG run-time option provides the optimum values to use when specifying the storage management 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 Language Environment storage options in order to achieve the space tuning purpose.)

Do not use the values from IGZTUNE when specifying the Language Environment tuning options. These values might not be optimal for tuning storage usage within the Language Environment environment. For recommended Language Environment storage option settings, see “Set recommended default Language Environment run-time options” on page 46.

Considerations for SPOUT output

Existing applications that specify the SPOUT run-time option will continue to run. The SPOUT option is mapped to the Language Environment RPTOPTS and RPTSTG run-time options.

The output from the RPTSTG and RPTOPTS reports is directed to the ddname specified in the MSGFILE run-time option. The default is SYSOUT. You can use the information generated by the RPTOPTS and RPTSTG options to determine the values to use when tuning storage with the Language Environment storage options.

Additional CICS considerations

This section lists additional considerations for COBOL programs run on CICS. It contains information on the following:

- Performance
- SORT interface change
- WORKING-STORAGE limits
- VS COBOL II NORENT programs
- IGZETUN or IGZEOPT and MSGFILE
- CICS HANDLE commands and the CBLPSHPOP run-time option
- DISPLAY statement

Performance considerations

Here are some actions you can take to maximize performance with Language Environment on CICS:

- Set the CICS SIT option RUWAPOOL to YES. Doing so can reduce the CICS GETMAIN/FREEMAIN activity when CICS LINK is used.
- Tune your Language Environment storage run-time options. Doing so can keep the CICS GETMAIN/FREEMAIN activity to a minimum, and avoids the overhead that occurs when an increment is needed.
- You might not want to use TERMTHDACT(DUMP) and TERMTHDACT(UADUMP) in production, because these TERMTHDACT sub-options can cause a lot of time to be spent writing Language Environment dump data to transient data queue CESE when a transaction abends.
- Do not run with run-time options RPTOPTS(ON) or RPTSTG(ON) in production. These options have a significant impact on performance.
- If you are getting any warning messages written by Language Environment to CESE, make changes so that they are not issued.

SORT interface change

When the SORT statement is used on CICS with the VS COBOL II run time, the SORT program is invoked using EXEC CICS LINK. When the SORT statement is used on CICS with Language Environment, the SORT program is loaded with EXEC CICS LOAD and is then invoked with a BASSM instruction.

WORKING-STORAGE limits

The WORKING-STORAGE limits for VS COBOL II program running on Language Environment are different than when running on the VS COBOL II run time as shown in Figure 33. This difference affects applications that are developed using Language Environment, and run in production using VS COBOL II.

Figure 33. WORKING-STORAGE limits for VS COBOL II programs on CICS

	WORKING-STORAGE limit	
DATA compiler option specification	**VS COBOL II run time**	**Language Environment run time**
DATA(24)	64K	Available space below the 16M line
DATA(31)	1M	128M

VS COBOL II NORENT programs

Although documented that VS COBOL II programs must be compiled with the RENT option to run on CICS, VS COBOL II Release 3.0 and later did not enforce this requirement. VS COBOL II APAR PN65736 added the function to diagnose attempts to run VS COBOL II NORENT programs on CICS (for VS COBOL II Release 3.0 and later).

Language Environment issues message IGZ0018S when it detects a VS COBOL II program compiled NORENT running on CICS. If you're developing on VS COBOL II without PN65736 applied, Language Environment will terminate your VS COBOL II applications run on CICS if they were compiled with NORENT.

IGZETUN or IGZEOPT and MSGFILE

When you run on Language Environment and have either IGZETUN or IGZEOPT link edited with your main program, Language Environment writes warning message IGZ0014W to the Language Environment MSGFILE each time a main program that contains IGZETUN or IGZEOPT is run. Writing the IGZ0014W message to the MSGFILE can cause a significant amount of additional system resource to be used. To suppress the writing of IGZ0014W, see “Using CEEWUCHA” on page 84.

CICS HANDLE commands and the CBLPSHPOP run-time option

The Language Environment CBLPSHPOP run-time option is used to control whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a VS COBOL II or IBM COBOL subroutine is called. Depending on the setting of CBLPSHPOP, you can get either faster CICS performance or compatible behavior (depending on how your programs are coded).

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

The VS COBOL II run time issues the PUSH HANDLE and POP HANDLE commands automatically on CALLs to COBOL subroutines. To receive this same behavior in Language Environment, you must specify the Language Environment CBLPSHPOP(ON) run-time option.

If your application calls COBOL (VS COBOL II or IBM COBOL) subroutines under CICS, performance is better with CBLPSHPOP(OFF) than with CBLPSHPOP(ON). (You can set the CBLPSHPOP run-time option on an enclave by enclave basis by using CEEUOPT.)

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 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 receive the same behavior as with the VS COBOL II run time, specify CBLPSHPOP(ON) if either of the following conditions are present:

1. The caller contains either of these CICS commands used for condition handling in CICS:
   - 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.

The next two examples show the effect of the CBLPSHPOP option on VS COBOL II compatibility.

EXAMPLE 1—no effect on compatibility

```
 IDENTIFICATION DIVISION.
 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.


 IDENTIFICATION DIVISION.
 PROGRAM-ID. PGM2
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  DATA1    PIC 9 VALUE 0.
 01  DATA2    PIC 9 VALUE 1.
 PROCEDURE DIVISION.
* Force a DIVIDE-BY-ZERO exception
     COMPUTE DATA1 = DATA2 / DATA1
     GOBACK.
```

In this example the setting of the CBLPSHPOP option will have no effect on VS COBOL II compatibility. If you specify CBLPSHPOP(ON), no HANDLE ABEND will be in effect for PGM2 since the CICS PUSH HANDLE performed by Language Environment will suspend the effects of the CICS HANDLE ABEND issued in PGM1. When the divide by zero occurs in PGM2, an ASRA abend occurs because there is no HANDLE ABEND active.

If you specify CBLPSHPOP(OFF), the divide by zero in PGM2 will cause CICS to ask Language Environment to branch to LBL1 in PGM1; however, Language Environment will not permit branches to labels across program boundaries. As with CBLPSHPOP(ON), the program will end with an ASRA abend.

EXAMPLE 2—effect on compatibility

```
 IDENTIFICATION DIVISION.
 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
* Force a DIVIDE-BY-ZERO exception
     COMPUTE DATA1 = DATA2 / DATA1
     EXEC CICS RETURN END-EXEC.
 LBL1.
     EXEC CICS RETURN END-EXEC.

 IDENTIFICATION DIVISION.
 PROGRAM-ID. PGM2
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 PROCEDURE DIVISION.
     EXEC CICS HANDLE ABEND LABEL(LBL1A) END-EXEC.
     GOBACK.
 LBL1A.
     GOBACK.
```

In this example the CBLPSHPOP option will affect compatibility with VS COBOL II. 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 ask Language Environment to branch to LBL1A in PGM2. Language Environment prevents this branch, and the program terminates with an ASRA abend.

DISPLAY statement

When you use the VS COBOL II run time on CICS, attempts to use the DISPLAY statement in a VS COBOL II program would cause a transaction abend.

When you use DISPLAY UPON SYSOUT (or just DISPLAY without specifying any UPON value) from a VS COBOL II or IBM COBOL program with Language Environment, the display output is written to the transient data queue CESE.

Chapter 8. Link-editing applications with Language Environment

This chapter describes the implications of link-editing OS/VS COBOL and VS COBOL II applications with Language Environment. When you link-edit applications with Language Environment, there might be a change in behavior, depending on if the application consist of:

- Programs compiled NORES
- Programs compiled RES

For additional information on relink-editing a COBOL load module with Language Environment, see Appendix K, “Link-edit example” on page 245.

Applications comprised of NORES programs

When you link-edit an application comprised of NORES programs with Language Environment, you need to:

- Link-edit all the programs in the load module with Language Environment
- Understand the changes in behavior that can occur

Normally, link-editing an OS/VS COBOL NORES application with Language Environment has no effect. The application will provide the same results as before, and cannot access Language Environment services.

However, in some cases, after link-editing an OS/VS COBOL NORES application with Language Environment, the application will exhibit “RES-behavior.” Figure 34 lists what can cause this change in behavior.

Figure 34. Program attributes causing changes in application behavior

Attribute	Comments
Contains a VS COBOL II program	Any applications with a VS COBOL II program will exhibit "RES-behavior" after link-editing the application with Language Environment.
Contains a COBOL/370 or IBM COBOL program	Any applications with a COBOL for OS/390 & VM, COBOL for MVS & VM, or COBOL/370 program will exhibit "RES-behavior" after link-editing the application with Language Environment. For additional information, see Chapter 16, "Adding IBM COBOL programs" on page 170.
Contains the IGZCBSN run-time routine	IGZCBSN is the COBOL/370 bootstrap routine.
Contains the IGZCBSO run-time routine	IGZCBSO is the COBOL for MVS & VM and COBOL for OS/390 & VM bootstrap routine.
Contains a program using an object module produced using the IGZBRDGE macro.	An object module produced by using the IGZBRDGE macro is normally used to convert static calls to dynamic calls. However, just the presence of an object module produced by using IGZBRDGE causes the behavior to change, regardless of how it is being used.

Note: For multiple load module applications, if the first load module contains one of the above attributes, it will behave as if it were RES. For a summary table of support for multiple load module applications running under Language Environment, see Figure 46 on page 174.

Implications of becoming RES-like: Only one of the attributes listed in Figure 34 on page 99 needs to be present to cause NORES applications to act as RES applications. If your NORES application does become "RES-like," you have additional considerations:

- Many of the considerations for programs compiled RES now apply to the NORES applications that you have link-edited with Language Environment (as indicated with the "NORES Linked" category of the margin icon at the beginning of each section in Chapter 5 and Chapter 6).
- Additional Language Environment services are available. For example, if you want to get a report of storage used by your program:
  - Before link-editing the NORES application with Language Environment, Language Environment was not initialized. You could not generate a storage report.
  - After link-editing the NORES application with Language Environment, Language Environment is initialized. You can now request a storage report by specifying the RPTSTG run-time option.
- Performance will be impacted.

Applications comprised of RES programs

When you link-edit an application comprised of RES programs with Language Environment, it will still have RES behavior.

VS COBOL II RES programs that are link-edited with Language Environment are enabled to use the Language Environment application-specific run-time options CSECT, CEEUOPT. Since IGZEOPT and CEEUOPT can exist in the same application, see Figure 29 on page 76 for details on which method will be in effect.

Chapter 9. Upgrading Language Environment release levels

Language Environment provides general object and load module compatibility for applications that ran with a previous release of Language Environment. In the following cases however, you must either relink or recompile when upgrading to specific Language Environment release levels:

- Calling between assembler and COBOL programs with different AMODEs under OS/390 V2R9 or later
- Running object-oriented COBOL applications under OS/390 Release 3
- Referencing symbolic feedback tokens
- Upgrading from Language Environment Release 1 - relink

Calling between assembler and COBOL under OS/390 V2R9 and later:
Changes were made in Language Environment Version 2 Release 9 that affect the AMODE upon return from a COBOL program to an assembler program. The changes were made in response to the problem reported in APAR PQ05151. The changes make Language Environment behave like the VS COBOL II run time, and provide consistent behavior regardless of the compiler used to compile the COBOL programs.

The change in behavior affects calls from assembler to COBOL only when there is a mode switch from AMODE(31) to AMODE(24) or from AMODE(24) to AMODE(31).

With Language Environment for OS/390 prior to Version 2 Release 9, the behavior for calls between COBOL and assembler is as follows:

- When a VS COBOL II or COBOL/370 subprogram returns to an assembler program caller, the AMODE is set based on the high-order bit in the R14 slot of the assembler program's save area (if the bit is on, control is returned in AMODE 31, otherwise control is returned in AMODE 24).
- When an OS/VS COBOL, COBOL for MVS & VM, or COBOL for OS/390 & VM subprogram returns to an assembler program caller, the AMODE is set to the same AMODE that was in effect when the COBOL program was entered.
- When using the COBOL reusable environment (RTEREUS, IGZERRE, or ILBOSTP0), and a COBOL program called by an assembler driver returns control to the assembler driver, the AMODE is set based on the high-order bit in the R14 slot of the assembler driver's save area (if the bit is on, control is returned in AMODE 31, otherwise control is returned in AMODE 24).

With Language Environment for OS/390 Version 2 Release 9 and later, the behavior for calls between COBOL and assembler is as follows:

- When a COBOL subprogram returns to an assembler program caller, the AMODE is set to the same AMODE that was in effect when the COBOL program was entered. Note that the behavior is the same regardless of which compiler was used.
- When using the COBOL reusable environment (RTEREUS, IGZERRE, or ILBOSTP0), and a COBOL program called by an assembler driver returns control to the assembler driver, the AMODE is set to the same AMODE that was in effect when the COBOL program was entered.

Running object-oriented COBOL applications under OS/390 Release 3: Object-oriented COBOL applications that are run under OS/390 Release 3 or later use SOM for OS/390 Release 3. Existing COBOL for MVS & VM object-oriented programs must be compiled with COBOL for OS/390 & VM when you move the programs to OS/390. For details, see “Determining which programs require upgrade” on page 159.

Referencing symbolic feedback tokens: Between Language Environment Release 3 and Release 4, changes were made to 11 condition tokens in CEEIGZCT. If your programs reference any of the following changed symbolic Feedback Tokens in CEEIGZCT you must recompile the programs when upgrading to Language Environment Release 4 or later:

CEE36U	CEE372	CEE58Q
CEE36V	CEE373	CEE58R
CEE37O	CEE374	CEE58S
CEE371	CEE375	

Language Environment Release 1 - upgrading IGZCBSN levels: When upgrading your run-time environment from LE/370 Release 1.0 to a later release, you must relink each COBOL/370 application load module that runs on CICS, to upgrade the IGZCBSN routine to a more current release.

The following sample job shows how to upgrade IGZCBSN:

```
//RELINK  EXEC PGM=IEWL,PARM='LIST,XREF,MAP',
//        COND=(4,LT)
//SYSPRINT DD  SYSOUT=*
//SYSLIB   DD  DISP=SHR,DSN=your.SCEELKED
//SYSLMOD  DD  DISP=SHR,DSN=your.new.loadlib
//PDSOLD   DD  DISP=SHR,DSN=your.old.loadlib
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLIN   DD  *
  REPLACE CEEARLU
  REPLACE CEEBETBL
  REPLACE CEEBINT
  REPLACE CEEBLLST
  REPLACE CEEBPIRA
  REPLACE CEEBTRM
  REPLACE CEESG005
  REPLACE CEESTART
  REPLACE IGZCBSN
  INCLUDE PDSOLD(myload)
  INCLUDE SYSLIB(CEESG005)
  ENTRY myentry
  NAME myload(R)
```

In the sample job above, tailor the lowercase letters to represent the particular programs being link-edited.

To replace all old COBOL library routines, see Appendix K, “Link-edit example” on page 245

Upgrading source programs

Chapter 10. Modifying OS/VS COBOL source programs

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

Terminology clarification

In this book, we use the term IBM COBOL as a general reference to:

- COBOL for MVS & VM Version 1 Release 2
- COBOL for OS/390 & VM Version 2 Release 1
- COBOL for OS/390 & VM Version 2 Release 2

Note: The language differences described in this chapter are a result of changes in the COBOL Standard. Thus, this information also applies to COBOL/370 programs.

IBM COBOL programs compiled with the NOCMPR2 compiler option provide 85 Standard-conforming programs, which is the strategic direction for the IBM COBOL language. When upgrading your OS/VS COBOL programs to IBM COBOL, it is recommended that you convert them to NOCMPR2 programs to take advantage of the language enhancements provided by the COBOL 85 Standard, as well as to position your company 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 OS/VS*
- *COBOL 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 15, “CICS conversion considerations—source” on page 160.
2. In the following sections, any reference to **COBOL 68 Standard** is a reference to the COBOL language supported by IBM Full American National Standard COBOL, Version 4 (Program 5734-CB2), or to LANGLVL(1) of OS/VS COBOL (Program 5740-CB1).
3. The information provided in this chapter, and throughout this manual, is intended for OS/VS COBOL Release 2.4, with the latest service updates applied.

Comparing OS/VS COBOL to IBM COBOL

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

Language elements that require change—quick reference

Figure 35 lists the language elements different in OS/VS COBOL and IBM COBOL. It also lists conversion tools, if any, available to automate the conversion.

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

- OS/VS COBOL language elements requiring other products
- OS/VS COBOL language elements not supported
- OS/VS COBOL language elements implemented differently
- Undocumented OS/VS COBOL extensions not supported

Figure 35 (Page 1 of 4). Language element differences between OS/VS COBOL and IBM COBOL

Language element	Conversion tool	Page
Abbreviated combined relation conditions		122
ACCEPT statement		122
ALPHABETIC class changes	CCCA	131
ALPHABET clause changes—ALPHABET key word	CCCA	131
Area A, periods in	CCCA	126
Arithmetic statement changes		131
ASSIGN ... OR	CCCA	114
ASSIGN TO *integer system-name*	CCCA	114
ASSIGN ... FOR MULTIPLE REEL /UNIT	CCCA	114
ASSIGN clause changes—*assignment-name* forms	CCCA	131
B symbol in PICTURE clause—changes in evaluation		132
BDAM file handling	CCCA[1]	113
BLANK WHEN ZERO clause and asterisk (*) override		122
CALL identifier statement—B symbol in PICTURE clause		132
CALL statement changes—procedure names and file names in USING phrase		132
CANCEL statement—B symbol in PICTURE clause		132
CLOSE...FOR REMOVAL statement		123
CLOSE statement—WITH POSITIONING, DISP phrases	CCCA	115
Combined abbreviated relation condition changes	CCCA	133
Comparing group to numeric packed-decimal item		123
COPY statement with associated names	CCCA	134
Communication feature		114
CURRENCY-SIGN clause changes—'/', '=', and 'L' characters		135
CURRENT-DATE special register	CCCA	115
DIVIDE...ON SIZE ERROR—change in intermediate results		140
Dynamic CALLs when running on CICS		123
Dynamic CALLs to programs with alternate entry without an intervening cancel		135
EXAMINE statement	CCCA	116
EXHIBIT statement	CCCA	116

Figure 35 (Page 2 of 4). Language element differences between OS/VS COBOL and IBM COBOL

Language element	Conversion tool	Page
EXIT PROGRAM/GOBACK statement changes		135
FILE STATUS clause changes	CCCA	135
FILE-LIMIT clause of the FILE-CONTROL paragraph	CCCA	117
Flow of control, no terminating statement		123
FOR MULTIPLE REEL /UNIT	CCCA	114
GIVING phrase of USE AFTER STANDARD ERROR declarative	CCCA	118
IF...OTHERWISE statement changes	CCCA	137
Index names—nonunique		124
INSPECT statement—in PROGRAM COLLATING SEQUENCE clause		141
IS as an optional word		140
ISAM file handling	CCCA	112
JUSTIFIED clause changes	CCCA	138
LABEL RECORDS clause with TOTALING/TOTALED AREA	CCCA	118
LABEL RECORD IS statement		124
MOVE statement—binary value and display value		124
MOVE statements and comparisons—scaling changes		138
MOVE CORRESPONDING statement	CCCA	125
MOVE statement—multiple TO specification		125
MOVE ALL—TO PIC 99		125
MOVE statement—warning message for numeric truncation		125
MULTIPLY...ON SIZE ERROR—change in intermediate results		140
Nonunique program-ID names	CCCA	127
NOTE statement	CCCA	118
Numeric class test on group items		139
Numeric data changes		139
Numeric-editing changes (Picture clause)		127
OCCURS clause (order of phrases)		126
OCCURS DEPENDING ON—ASCENDING/DESCENDING KEY phrase		139
OCCURS DEPENDING ON—value for receiving items changed	CCCA	139
ON statement	CCCA	118
ON SIZE ERROR phrase—changes in intermediate results		140
OPEN statement failing for VSAM files (file status 39)		119
OPEN statement failing for QSAM files (file status 39)		119
OPEN statement with LEAVE, REREAD, and DISP phrases	CCCA	119
OPEN REVERSED statement		126
OTHERWISE clause changes		137
Paragraph names not allowed as parameters		132
PERFORM statement—changes in the VARYING/AFTER phrases		141

Figure 35 (Page 3 of 4). Language element differences between OS/VS COBOL and IBM COBOL

Language element	Conversion tool	Page
PERFORM statement—second UNTIL		126
Periods, consecutive in any division		126
Periods in Area A	CCCA	126
Periods missing on paragraphs	CCCA	127
Periods missing at the end of SD, FD, or RD		127
PICTURE clause (numeric-editing changes)		127
PROGRAM COLLATING SEQUENCE clause changes		141
Program-ID names, nonunique	CCCA	127
Qualification - using the same phrase repeatedly		127
READ statement—redefined record keys in the KEY phrase		127
READ and RETURN statement changes—INTO phrase		141
READY TRACE and RESET TRACE statements	CCCA	119
RECORD CONTAINS n CHARACTERS clause		127
RECORD KEY phrase and ALTERNATE RECORD KEY phrase		128
REDEFINES clause in SD or FD entries	CCCA	128
REDEFINES clause with tables		128
Relation conditions	CCCA	128
REMARKS paragraph	CCCA	120
RENAMES clause—nonunique, nonqualified data names		128
Report Writer statements	Report Writer Precompiler	111
RERUN clause changes		142
RESERVE clause changes	CCCA	142
Reserved Word list changes	CCCA	142
SEARCH statement changes	CCCA	142
Segmentation changes—PERFORM statement in independent segments		143
SELECT statement without a corresponding FD		128
SELECT OPTIONAL clause changes	CCCA	143
SORT special registers		143
SORT verb		129
SORT/MERGE		129
Source language debugging changes		144
START ... USING KEY statement	CCCA	120
STRING statement—in PROGRAM COLLATING SEQUENCE clause		141
STRING statement—sending field identifier		129
Subscripts out of range—flagged at compile-time		144
THEN as a statement connector	CCCA	121
TIME-OF-DAY special register	CCCA	121

Figure 35 (Page 4 of 4). Language element differences between OS/VS COBOL and IBM COBOL

Language element	Conversion tool	Page
TOTALING/TOTALED AREA phrases in LABEL RECORDS clause	CCCA	118
TRANSFORM statement	CCCA	121
UNSTRING statement—in PROGRAM COLLATING SEQUENCE clause		141
UNSTRING statement—coding with 'OR', 'IS', or a numeric edited item	CCCA	130
UNSTRING statement—multiple INTO phrases		130
UNSTRING statements—subscript evaluation changes		144
UPSI switches	CCCA	146
USE AFTER STANDARD ERROR—GIVING phrase	CCCA	118
USE BEFORE STANDARD LABEL statement	CCCA	122
VALUE clause—signed value in relation to the PICTURE clause	CCCA	130
VALUE clause—condition names	CCCA	146
WHEN-COMPILED special register	CCCA	146
WRITE AFTER POSITIONING statement	CCCA	146

Note:

1 This is a partial conversion.

Using conversion tools to convert programs to COBOL 85 Standard

To help you make changes needed when upgrading to IBM COBOL NOCMPR2 you can use any of the following:

- The COBOL conversion tool (CCCA)
- The OS/VS COBOL MIGR compiler option
- The IBM COBOL FLAGMIG and CMPR2 compiler options

A brief description of these conversion tools follows. See Appendix C, “Conversion tools for source programs” on page 199 for additional information.

Note: Non-IBM tools are also available to help automate the conversion to the COBOL 85 Standard. For details, see “Vendor products” on page 208.

COBOL Conversion Tool (CCCA)

The COBOL and CICS/VS Command Level Conversion Aid (CCCA) is **not** for CICS only; it converts any old COBOL to IBM COBOL. The CCCA provides you with either a report of the statements that need to be changed or the actual converted program itself.

For details, see “COBOL and CICS/VS Command Level Conversion Aid (CCCA)” on page 204 and the *COBOL and CICS/VS Command Level Conversion Aid Program Description and Operations Manual*.

OS/VS COBOL MIGR compiler option

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

"MIGR compiler option" on page 199 lists the items flagged by MIGR. A complete description of MIGR-flagged items is included in Appendix H of *IBM VS COBOL for OS/VS*.

CMPR2 and FLAGMIG compiler options

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

Language elements requiring other products for support

Although some OS/VS COBOL language elements are not supported in IBM COBOL, you can get equivalent function by using other products.

Report Writer

The Report Writer feature is supported through use of the Report Writer Precompiler. In order for existing Report Writer code to work with IBM COBOL, 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 IBM COBOL compiler, your Report Writer applications can run above the 16M line. Through IBM COBOL, you can also extend their processing capabilities.

This method requires the use of both the Report Writer Precompiler and the IBM COBOL 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 IBM COBOL 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 OS/VS COBOL-compiled Report Writer programs under Language Environment: You can run existing OS/VS COBOL Report Writer applications using Language Environment without compiling with IBM COBOL. For details

on running existing OS/VS COBOL programs using the Language Environment run-time library, see Chapter 6, “Moving from the OS/VS COBOL run time” on page 58. To compile OS/VS COBOL applications with Report Writer statements, you must continue to use the OS/VS COBOL compiler.

OS/VS COBOL Report Writer programs will not run above the 16M line.

Report Writer language items affected: The Report Writer language items no longer accepted by IBM COBOL 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 C, “Conversion tools for source programs” on page 199.

Language elements no longer implemented

The following OS/VS COBOL language elements are not supported by IBM COBOL:

- ISAM file handling
- BDAM file handling
- Communication feature

With IBM COBOL, support for most of the COBOL 68 Standard language elements has been removed. There are also miscellaneous OS/VS COBOL language items that are not implemented in IBM COBOL.

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

ISAM file handling

IBM COBOL does not support the processing of ISAM files. Convert any ISAM files to Virtual Storage Access Method/Keyed Sequential Data Set (VSAM/KSDS) files.

ISAM file handling language items affected: The ISAM language items no longer accepted by IBM COBOL 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

Automated conversion options: Two conversion tools can help you convert ISAM files to VSAM/KSDS files. You can use either IDCAMS REPRO or CCCA, depending on the design of your application. The IDCAMS REPRO facility will perform the conversion unless the file has a hardware dependency.

The COBOL conversion tool (CCCA) can automatically convert the file definition and I/O statements from your ISAM COBOL language to VSAM/KSDS COBOL language. The CCCA conversion tool is described in Appendix C, “Conversion tools for source programs” on page 199.

Manual conversion actions: 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 OS/VS COBOL compiler. You can then separate the rest of the application logic into programs that can be upgraded to IBM COBOL.

You can then run your application consisting of both OS/VS COBOL programs and IBM COBOL programs under Language Environment. For details on running existing programs under Language Environment, see:

Chapter 6, “Moving from the OS/VS COBOL run time” on page 58
Chapter 16, “Adding IBM COBOL programs” on page 170

Note: This method is presented only as a short-term migration path. Because OS/VS COBOL is no longer in service, you should rewrite your programs to eliminate the dependency on OS/VS COBOL and ISAM. (If you do not mind using an unsupported compiler, this method can still work for you.)

BDAM file handling

IBM COBOL does not support the processing of BDAM files. Convert any BDAM files to Virtual Storage Access Method/Relative Record Data Set (VSAM/RRDS) files.

BDAM file handling language items affected: The BDAM language items no longer accepted by IBM COBOL are:

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

Automated conversion options: The COBOL conversion tool (CCCA) can automatically convert your BDAM COBOL language to VSAM/RRDS COBOL language. You must provide the key algorithm. The CCCA conversion tool is described in Appendix C, “Conversion tools for source programs” on page 199.

Non-IBM tools are also available to convert a BDAM file to a VSAM/RRDS file. For details, see “Vendor products” on page 208.

Manual conversion actions: If the design of your application makes it impossible to convert to VSAM, you can restructure the application to separate the BDAM statements into an I/O program that can be compiled by the OS/VS COBOL com-

piler. You can then separate the rest of the application logic into programs that can be upgraded to IBM COBOL.

You can then run your application consisting of both OS/VS COBOL programs and IBM COBOL programs under Language Environment. For details on running existing programs under Language Environment, see:

Chapter 6, “Moving from the OS/VS COBOL run time” on page 58
Chapter 16, “Adding IBM COBOL programs” on page 170

Note: This method is presented only as a short-term migration path. Because OS/VS COBOL is no longer in service, you should rewrite your programs to eliminate the dependency on OS/VS COBOL and BDAM. (If you do not mind using an unsupported compiler, this method can still work for you.)

Communication feature

The Communication feature is not supported by IBM COBOL.

Communication language items affected

The Communication language items not accepted by IBM COBOL are:

ACCEPT MESSAGE COUNT statement
COMMUNICATION SECTION
DISABLE statement
ENABLE statement
RECEIVE statement
SEND statement

Communication conversion actions

Existing TCAM applications that use the OS/VS COBOL SEND and RECEIVE statements run under Language Environment with one exception: the QUEUE run-time option of OS/VS COBOL is not supported. (The QUEUE run-time option is used only in an OS/VS COBOL program with a RECEIVE statement in a CD ... FOR INITIAL INPUT.)

For more information, see *IBM VS COBOL for OS/VS*, and *IBM OS/VS COBOL Compiler and Library Programmer's Guide*.

Language elements not supported

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

ASSIGN ... OR
OS/VS COBOL accepted the ASSIGN ... OR clause. To use this clause under IBM COBOL, you must remove the OR.

ASSIGN TO *integer system-name*
OS/VS COBOL accepted the ASSIGN TO *integer system-name* clause. To use this clause under IBM COBOL, you must remove the integer.

ASSIGN ... FOR MULTIPLE REEL/UNIT
OS/VS COBOL accepted the ASSIGN ... FOR MULTIPLE REEL/UNIT phrase, and treated it as documentation. IBM COBOL does not support this phrase.

CLOSE statement—WITH POSITIONING, DISP phrases

OS/VS COBOL accepted the WITH POSITIONING and DISP phrases of the CLOSE statement provided as IBM extensions in OS/VS COBOL. In IBM COBOL, these phrases are not accepted.

CURRENT-DATE special register

OS/VS COBOL accepted 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)
```

IBM COBOL 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, you must change an OS/VS COBOL program with statements similar to the following:

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

An example of one way to change it, keeping the two-digit year format, 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.
```

An example of how to change it and specify a four-digit year 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(04).

01  CURRENT-DATE.
    02 YEAR              PIC X(04).
    02 MONTH-OF-YEAR     PIC X(02).
    02 DAY-OF-MONTH      PIC X(02).
  ⋮
    MOVE FUNCTION CURRENT-DATE(1:8) TO CURRENT-DATE.
    MOVE CORRESPONDING CURRENT-DATE TO DATE-IN-PROGRAM.
```

EXAMINE statement

OS/VS COBOL accepted the EXAMINE statement.

IBM COBOL does not accept the EXAMINE statement.

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

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

Replace it in IBM COBOL 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

OS/VS COBOL accepted the EXHIBIT statement.

IBM COBOL does not accept the EXHIBIT statement.

In IBM COBOL, 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:

```
    OS/VS COBOL                          IBM COBOL

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

```
    OS/VS COBOL                          IBM COBOL

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

```
    OS/VS COBOL                          IBM COBOL

WORKING-STORAGE SECTION.          WORKING-STORAGE SECTION.
77  DAT-1 PIC X(8).              77  DAT-1    PIC X(8).
77  DAT-2 PIC X(8).              77  DAT-2    PIC X(8).
                                  77  DAT1-CMP PIC X(8).
                                  77  DAT2-CMP PIC X(8).
        .                                 .
        .                                 .
        .                                 .
  EXHIBIT CHANGED NAMED            IF DAT-1 NOT EQUAL TO DAT1-CMP
      DAT-1 DAT-2                      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

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

GIVING phrase of USE AFTER STANDARD ERROR declarative

In OS/VS COBOL, you could specify the GIVING phrase of the USE AFTER STANDARD ERROR declarative. IBM COBOL does not support this phrase. Therefore, you must remove any occurrences of the GIVING phrase of the USE AFTER STANDARD ERROR declarative.

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

LABEL RECORDS clause with TOTALING/TOTALED AREA phrases

OS/VS COBOL allowed the TOTALING and TOTALED phrases of the LABEL RECORDS clause.

IBM COBOL does not support these phrases. Therefore, you must remove any occurrences of the TOTALING/TOTALED phrases from the LABEL RECORDS clause. Also check the variables associated with these phrases.

To receive similar functions with COBOL for OS/390 & VM, you must:

- Remove references to TOTALING or TOTALED from the LABEL RECORDS clause of the FD. Define equivalent fields in WORKING-STORAGE.

  In the equivalent fields, save any key or record data (for TOTALED AREA) or update and save any record count (for TOTALING AREA) before each WRITE to the file (this reflects the last record actually written to particular volume). Do **not** save this counter in the record area of the FD.
- Include “RESERVE 1 AREA” in the SELECT clause.
- Allow for only 1 record per block in either the FD or the overriding JCL.
- Include:

  ```
  DECLARATIVES.
        USE AFTER STANDARD ENDING FILE (or REEL)
         LABEL PROCEDURE ON filename.
  ```

  Then, either format labels for OUTPUT files or save any data from INPUT file labels (AFTER STANDARD BEGINNING).
- Specify the NOAWO compiler option.
- Do not specify the DD DCB option OPTCD=T. It is not supported and results are unpredictable.

NOTE statement

OS/VS COBOL accepted the NOTE statement. IBM COBOL does not accept the NOTE statement. Therefore, for IBM COBOL delete all NOTE statements and use comment lines instead for the entire NOTE paragraph.

ON statement

OS/VS COBOL accepted the ON statement. IBM COBOL does not accept the ON statement.

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

OPEN statement failing for QSAM files (file status 39)

In OS/VS COBOL, the fixed file attributes for QSAM files did not need to match your COBOL program or JCL. In IBM COBOL, if the following do not match, an OPEN statement in your program might not execute successfully:

- The fixed file attributes specified in the DD statement or the data set 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 I, “Preventing file status 39 for QSAM files” on page 241.

OPEN statement failing for VSAM files (file status 39)

In OS/VS COBOL, the RECORDSIZE defined in your VSAM files associated with IDCAMS was not required to match your COBOL program. In IBM COBOL they must match. The following rules apply to VSAM ESDS, KSDS, RRDS, and VRRDS file definitions:

Figure 36. 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.

OPEN statement with the LEAVE, REREAD, and DISP phrases

OS/VS COBOL allowed the OPEN statement with the LEAVE, REREAD and DISP phrases. IBM COBOL does not allow these phrases.

To replace the REREAD function, define a copy of your input records in the WORKING-STORAGE SECTION and move each record into WORKING-STORAGE after it is read.

READY TRACE and RESET TRACE statements

OS/VS COBOL allowed the READY TRACE and RESET TRACE statements. IBM COBOL does not support these statements.

To get function similar to the READY TRACE statement, you can use either the Debug Tool, or the COBOL language available in the IBM COBOL compiler.

If using the Debug Tool, compile your program with the TEST(ALL,SYM) option and use the following Debug Tool command:

```
"AT GLOBAL LABEL PERFORM;
 LIST LINES %LINE; GO; END-PERFORM;"
```

If using the COBOL language, the IBM COBOL 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 0.
     88  READY-TRACE           VALUE 1.
     88  RESET-TRACE           VALUE 0.
  ⋮
PROCEDURE DIVISION.
  DECLARATIVES.
  COBOL-II-DEBUG SECTION.
    USE FOR DEBUGGING ON ALL PROCEDURES.
  COBOL-II-DEBUG-PARA.
    IF READY-TRACE THEN
        DISPLAY DEBUG-NAME
    END-IF.
  END DECLARATIVES.
  MAIN-PROCESSING SECTION.
  ⋮
  PARAGRAPH-3.
  ⋮
    SET READY-TRACE TO TRUE.
  PARAGRAPH-4.
  ⋮
  PARAGRAPH-6.
  ⋮
    SET RESET-TRACE TO TRUE.
  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

OS/VS COBOL accepted the REMARKS paragraph.

IBM COBOL does not accept the REMARKS paragraph. As a replacement, use comment lines beginning with an * in column 7.

START...USING KEY statement

OS/VS COBOL allowed the START statement with the USING KEY phrase. IBM COBOL does not. Instead, for IBM COBOL, you can specify the START statement with the KEY IS phrase.

THEN as a statement connector

OS/VS COBOL accepted the use of THEN as a statement connector.

The following example shows the OS/VS COBOL usage:

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

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

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

TIME-OF-DAY special register

OS/VS COBOL supported the TIME-OF-DAY special register. It was valid only as the sending field in a MOVE statement. TIME-OF-DAY had the following 6-byte EXTERNAL decimal format:

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

IBM COBOL does not support the TIME-OF-DAY special register.

Therefore, you must change an OS/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:

```
MOVE FUNCTION CURRENT-DATE (9:6) TO TIME-IN-PROGRAM
```

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

TRANSFORM statement

OS/VS COBOL supported the TRANSFORM statement. IBM COBOL does not support the TRANSFORM statement, but it does support the INSPECT statement. Therefore, any TRANSFORM statements in your OS/VS COBOL program must be replaced by INSPECT CONVERTING statements.

For example, in the following OS/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 IBM COBOL):

```
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

OS/VS COBOL accepted the USE BEFORE STANDARD LABEL statement. IBM COBOL does not support this statement.

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

Undocumented OS/VS COBOL extensions not supported

This section consists primarily of COBOL statements that are not flagged by the MIGR option. These statements were accepted by the OS/VS COBOL compiler; some are not accepted by IBM COBOL.

Because these language elements are undocumented extensions to OS/VS COBOL, they are not considered to be valid OS/VS COBOL code. This list might not contain all undocumented extensions; it includes all of the undocumented extensions of which we are aware.

Abbreviated combined relation conditions and use of parentheses

OS/VS COBOL accepted the use of parentheses within an abbreviated combined relation condition.

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

- Within the scope of an abbreviated combined relation condition, IBM COBOL does not support relational operators inside parentheses. For example:

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

- Some incorrect usages of parentheses in relation conditions were accepted by OS/VS COBOL, but are not by IBM COBOL. For example:

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

ACCEPT statement

OS/VS COBOL accepted the ACCEPT statement without the keyword FROM between the identifier and the mnemonic or function name.

IBM COBOL does not accept such an ACCEPT statement.

BLANK WHEN ZERO clause and asterisk (*) override

In OS/VS COBOL, if you specified the BLANK WHEN ZERO clause and the asterisk (*) as a zero suppression symbol for the same entry, zero suppression would override BLANK WHEN ZERO.

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

If you have specified both the BLANK WHEN ZERO clause and the asterisk as a zero suppression symbol in your OS/VS COBOL programs,

to get the same behavior in IBM COBOL, remove the BLANK WHEN ZERO clause.

CLOSE...FOR REMOVAL statement

OS/VS COBOL allowed the FOR REMOVAL clause for sequential files, and it had an effect on the execution of the program. IBM COBOL syntax-checks the statement but it has no effect on the execution of the program.

Comparing group to numeric packed-decimal item

OS/VS COBOL allowed a comparison between a group and a numeric packed-decimal item, but generated code that produced an incorrect result.

For example, the result of the comparison below is the message

"1 IS NOT > 0"

and is not the numerically correct

"1 > 0"

```
    05  COMP-TABLE.
        10 COMP-PAY              PIC 9(4).
        10  COMP-HRS             PIC 9(3).
    05  COMP-ITEM                PIC S9(7) COMP-3.

PROCEDURE DIVISION.
    MOVE 0 TO COMP-PAY COMP-HRS.
    MOVE 1 TO COMP-ITEM.
    IF COMP-ITEM > COMP-TABLE
       DISPLAY '1 > 0'
    ELSE
       DISPLAY '1 IS NOT > 0'.
```

IBM COBOL does not allow such a comparison.

Dynamic CALLs when running on CICS

Although dynamic CALLs from OS/VS COBOL programs were never supported when running under CICS, OS/VS COBOL did not diagnose programs that issued them.

If your OS/VS COBOL programs issue dynamic CALLs when running on CICS under Language Environment, the programs will abend with abend code U3504.

IBM COBOL programs can make dynamic CALLs to other IBM COBOL programs or even to PL/I and C/C++ programs under CICS.

Flow of control, no terminating statement

In OS/VS COBOL, it would be possible to link-edit an assembler program to the end of an OS/VS COBOL program and have the flow of control go from the end of the COBOL program to the assembler program.

In IBM COBOL, if you do not code a terminating statement at the end of your program (STOP RUN or GOBACK), the program will terminate with an implicit GOBACK. The flow of control cannot go beyond the end of the COBOL program.

If you have programs that rely on 'falling through the end' into another program, change the code to a CALL interface to the other program.

Index names

OS/VS COBOL allowed the use of qualified index names.

IBM COBOL does not allow qualified index names; index names must be unique if referenced.

LABEL RECORD IS statement

OS/VS COBOL accepted a LABEL RECORD clause without the word RECORD. You could have LABEL IS OMITTED instead of LABEL RECORD IS OMITTED.

IBM COBOL does not accept such a LABEL RECORD clause.

MOVE statement - binary value and display value

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

This example actually shows invalid COBOL coding, since 10 digits are being moved into a 9-digit item.

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

	OS/VS COBOL NOTRUNC	IBM COBOL TRUNC(OPT)
Binary value	x'499602D2'	x'0DFB38D2'
Display value	234567890	234567890

For OS/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 IBM COBOL, 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 OS/VS COBOL and by IBM COBOL when compiled with TRUNC(OPT).

MOVE CORRESPONDING statement

- OS/VS COBOL allowed more than one receiver with MOVE CORRESPONDING. IBM COBOL does not. Therefore, you must change the following OS/VS COBOL statement:

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

  to two IBM COBOL MOVE CORRESPONDING statements:

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

- Releases prior to Release 2.4 of OS/VS COBOL accepted nonunique subordinate data items in the receiver of a MOVE CORRESPONDING statement. IBM COBOL does not. For example:

  ```
  01  KANCFUNC.
      03  CL  PIC XX.
      03  KX9 PIC XX.
      03  CC  PIC XX.

  01  HEAD1-AREA.
      03  CL  PIC XX.
      03  KX9 PIC XX.
      03  CC  PIC XX.
      03  KX9 PIC XX.
  ⋮
      MOVE CORR KANCFUNC to HEAD1-AREA.
  ```

  For IBM COBOL, change the receiver to have unique names.

MOVE statement—multiple TO specification

OS/VS COBOL allowed the reserved word TO to precede each receiver in a MOVE statement. For example:

```
MOVE aa TO bb TO cc
```

IBM COBOL does not. The above statement must be changed to:

```
MOVE aa TO bb cc
```

MOVE ALL—TO PIC 99

OS/VS COBOL allowed group moves into a fixed numeric receiving field. For example:

```
MOVE ALL ' ' TO num1
```

where, `num1` is PIC 99.

IBM COBOL does not.

MOVE statement—warning message for numeric truncation

OS/VS COBOL issued a warning message for a MOVE statement with a numeric receiver that would result in a loss of digits. For example:

```
77 A PIC 999.
77 B PIC 99.
⋮
    MOVE A TO B.
```

VS COBOL II, COBOL for MVS & VM, and COBOL for OS/390 & VM Version 2 Release 1 do not issue a warning message for this case.

COBOL for OS/390 & VM Version 2 Release 2 issues a warning message if the new compiler option DIAGTRUNC is in effect.

OCCURS clause

OS/VS COBOL allowed nonstandard order for phrases following the OCCURS clause.

For example, the following code sequence would be allowed in OS/VS COBOL:

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

IBM COBOL does not allow a nonstandard order of phrases following OCCURS. For IBM COBOL, the example must be changed to the following:

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

OPEN REVERSED statement

OS/VS COBOL accepted the REVERSED phrase for multireel files.

IBM COBOL does not.

PERFORM statement—second UNTIL

OS/VS COBOL allowed a second UNTIL in a PERFORM statement, as in the following example:

```
PERFORM CHECK-FOR-MATCH THRU CHECK-FOR-MATCH-EXIT
    UNTIL PARM-COUNT = 7
    OR SSREJADV-EOF.
```

IBM COBOL does not allow a second UNTIL statement. It must be removed.

Periods, Area A

OS/VS COBOL allowed you to code a period in Area A following an invalid Area-A item (or no item). With IBM COBOL, a period in Area A must be preceded by a valid Area-A item.

Periods, consecutive in any division

OS/VS COBOL allowed you to code two consecutive periods in any division.

IBM COBOL issues a warning message (RC = 4) if two periods in a row are found in the PROCEDURE DIVISION, and a severe message (RC = 12) if two periods in a row are found in either the ENVIRONMENT DIVISION or the DATA DIVISION.

The following would be accepted by OS/VS COBOL, but would receive a severe (RC = 12) error and a warning (RC = 4) under IBM COBOL:

```
WORKING-STORAGE SECTION.
01 A PIC 9. .
⋮
    MOVE 1 TO A. .
  ⋮
    GOBACK.
```

Periods missing at the end of SD, FD, or RD

A period is required at the end of a Sort, File, or Report Description, preceding the 01-level indicator.

OS/VS COBOL diagnosed the missing period with a warning message (RC = 4).

IBM COBOL issues an error message (RC = 8).

Periods missing on paragraphs

Releases prior to Release 2.4 of OS/VS COBOL accepted paragraph names not followed by a period. Release 2.4 of OS/VS COBOL issued a warning message (RC = 4).

IBM COBOL issues an error message (RC = 8).

PICTURE string

OS/VS COBOL accepted a PICTURE string with all Z's to the left of the implied decimal point, a Z immediately to the right of the implied decimal point, but ending with a 9 or 9–. For example:

```
05 WEIRD-NUMERIC-EDITED PIC Z(11)VZ9.
```

IBM COBOL does not. You must change the `Z9` to either `ZZ` or `99`.

PROGRAM-ID names, nonunique

OS/VS COBOL allowed a data-name or paragraph-name to be the same as the PROGRAM-ID name. IBM COBOL requires the PROGRAM-ID name to be unique.

Qualification—using the same phrase repeatedly

```
A of B of B
```

OS/VS COBOL allowed repeating of phrases.

IBM COBOL does not.

READ statement—redefined record keys in the KEY phrase

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

IBM COBOL 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 an OS/VS COBOL program was overridden if an OCCURS DEPENDING ON clause was specified in the FD, and produced a file containing **variable**-length records instead of **fixed**-length records.

Under IBM COBOL, the RECORD CONTAINS n CHARACTERS clause produces a file containing **fixed**-length records.

RECORD KEY phrase and ALTERNATE RECORD KEY phrase

OS/VS COBOL allowed the leftmost character position of the ALTERNATE RECORD KEY data-name-4 to be the same as the leftmost character position of the RECORD KEY or of any other ALTERNATE RECORD KEY phrases.

IBM COBOL does not allow this.

REDEFINES clause in SD or FD entries

Releases prior to Release 2.4 of OS/VS COBOL accepted a REDEFINES clause in a level-01 SD or FD. IBM COBOL and Release 2.4 of OS/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 IBM COBOL, delete the REDEFINES clause.

REDEFINES clause with tables

OS/VS COBOL allowed you to specify tables within the REDEFINES clause. For example, OS/VS COBOL would issue a warning message (RC = 4) for the following example:

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

IBM COBOL does not allow tables to be redefined, and issues a severe (RC = 12) message for the example above.

Relation conditions

Releases prior to Release 2.4 of OS/VS COBOL accepted invalid operators in relation conditions. The following table lists the operators accepted by OS/VS COBOL Release 2.3 that are not accepted by IBM COBOL. It also shows the valid coding for IBM COBOL programs.

OS/VS COBOL R2.3	IBM COBOL
= TO	= or EQUAL TO
> THAN	> or GREATER THAN
< THAN	< or LESS THAN

RENAMES clause—nonunique, nonqualified data names

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

SELECT statement without a corresponding FD

OS/VS COBOL accepted a SELECT statement that does not have a corresponding FD entry. IBM COBOL does not accept this.

SORT verb

1. At early maintenance levels, the OS/VS COBOL compiler accepted the UNTIL and TIMES phrases in the SORT verb, for example:

```
SORT FILE-1
 ON ASCENDING KEY AKEY-1
  INPUT PROCEDURE IPROC-1
   OUTPUT PROCEDURE OPROC-1
    UNTIL AKEY-1 = 99.


SORT FILE-2
 ON ASCENDING KEY AKEY-2
  INPUT PROCEDURE IPROC-2
   OUTPUT PROCEDURE OPROC-2
    10 TIMES.
```

   IBM COBOL does not.

2. In a SORT statement, the correct syntax allows ASCENDING KEY or DESCENDING KEY followed by a data-name which is the sort key. The word KEY is optional.

   OS/VS COBOL accepted IS if used following ASCENDING KEY. For example:

```
SORT SORT-FILE
     ASCENDING KEY IS SD-NAME-FIELD
     USING INPUT-FILE
     GIVING SORTED-FILE.
```

SORT/MERGE

With OS/VS COBOL, a MOVE to the SD buffer before the first RETURN in a SORT or MERGE output PROCEDURE did not overlay the data of the first record.

With IBM COBOL such a MOVE would overlay the data of the first record. During a SORT or MERGE operation, the SD data item is used. You must not use it in the OUTPUT PROCEDURE before the first RETURN statement executes. If data is moved into this record area before the first RETURN statement, the first record to be returned will be overwritten.

STRING statement—sending field identifier

OS/VS COBOL allowed a numeric sending field identifier that is not an integer. Under IBM COBOL, a numeric sending field identifier must be an integer.

UNSTRING statement—coding with 'OR', 'IS', or a numeric edited item

OS/VS COBOL would not issue a diagnostic error message for UNSTRING statements containing any of the following instances of invalid coding:

1. Lack of the required word “OR” between literal-1 and literal-2, 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.
```

3. Use of a numeric edited item as the source of an UNSTRING statement, as in:

```
01 NUM-ED-ITEM    PIC $$9.99+
  ⋮
    UNSTRING NUM-ED-ITEM DELIMITED BY '$'
         INTO RECV-FIELD-1
         POINTER PTR-FIELD
```

   IBM COBOL only allows nonnumeric data items as senders in the UNSTRING statement.

IBM COBOL issues a message if an UNSTRING statement containing any of these errors is encountered.

UNSTRING statement—multiple INTO phrases

OS/VS COBOL issued a warning (RC = 4) message when multiple INTO phrases were coded. For example:

```
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-R2 DELIMITER IN ID-D3 COUNT IN ID-C3
```

IBM COBOL does not allow multiple INTO phrases in an UNSTRING statement.

VALUE clause—signed value in relation to the PICTURE clause

In OS/VS COBOL, the VALUE clause literal could be signed if the PICTURE clause was unsigned.

In IBM COBOL, the VALUE clause literal must match the PICTURE clause and the sign must be removed.

Language elements changed from OS/VS COBOL

The following OS/VS COBOL language elements are changed in IBM COBOL in order to conform to the COBOL 85 Standard. For some elements, the syntax of the language is different. For others, the language syntax is unchanged, but the execution results can be different (semantics have changed).

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 OS/VS COBOL, only uppercase letters and the space character were considered to be ALPHABETIC.

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

If your OS/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 IBM COBOL ALPHABETIC-UPPER class test for the OS/VS COBOL ALPHABETIC test.

ALPHABET clause changes—ALPHABET key word

In OS/VS COBOL, the key word ALPHABET was not allowed in the ALPHABET clause.

In IBM COBOL, the ALPHABET key word is required.

Arithmetic statement changes

IBM COBOL supports the following arithmetic items with enhanced accuracy:

- Usage of floating-point data items
- Usage of floating-point literals
- Usage of fractional exponentiation

Therefore, for arithmetic statements that contain these items, IBM COBOL might provide more accurate results than OS/VS COBOL. You will need to test your applications to verify that these changes do not have a negative impact on them.

ASSIGN clause changes

IBM COBOL supports only the following format of the ASSIGN clause:

```
ASSIGN TO assignment-name
```

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

QSAM files	[comments-][S-]name
VSAM sequential files	[comments-][AS-]name
VSAM indexed or relative files	[comments-]name
LINE SEQUENTIAL files	[comments-]name

If your OS/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 format supported by IBM COBOL.

B symbol in PICTURE clause—changes in evaluation

OS/VS COBOL accepted the PICTURE symbols A and B in definitions for alphabetic items.

IBM COBOL 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 OS/VS COBOL and IBM COBOL for evaluations of the:

- CANCEL statement
- CALL statement
- Class test
- STRING statement

CALL statement changes

OS/VS COBOL accepted paragraph names, section names, and file names in the USING phrase of the CALL statement.

IBM COBOL CALL statements do not accept procedure names and only accept QSAM file names in the USING phrase. Therefore, you must remove the procedure names and make sure that file names used in the USING phrase of the CALL statement name QSAM physical sequential files.

To convert OS/VS COBOL programs that CALL assembler programs passing procedure names, you will need to rewrite the assembler routines. In OS/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 IBM COBOL, code your assembler routines so that they return to the point of origin with an assigned number. 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 in OS/VS COBOL:

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

In IBM COBOL:

```
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.

Many COBOL users coded assembler programs using the 390 SPIE mechanism to get control when there is an error or condition. These routines can pass control to a COBOL program at a paragraph whose name was passed to the SPIE routine. Applications that use these user-written SPIE routines should be converted to use Language Environment condition handling.

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: OS/VS COBOL with LANGLVL(1) accepted 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
  ```

OS/VS COBOL with LANGLVL(2) and IBM 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 when moving from OS/VS COBOL with LANGLVL(1), you should expand all abbreviated combined conditions to their full unabbreviated forms.

Parenthesis evaluation: OS/VS COBOL accepted the use of parentheses within an abbreviated combined relational condition.

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

- Within the scope of an abbreviated combined relation condition, IBM COBOL does not support relational operators inside parentheses. For example:

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

- Some incorrect usages of parentheses in relation conditions were accepted by OS/VS COBOL, but are not accepted by IBM COBOL. For example:

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

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

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

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

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

Note: IBM COBOL 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 with associated names

OS/VS COBOL with LANGLVL(1) allowed COPY statements to be preceded by an 01-level indicator, which would result in the 01-level name replacing the 01-level name in the COPY member. For example, with the following contents of COPY member MBR-A:

```
01 RECORD-A.
   05 FIELD-A...
   05 FIELD-B...
```

and a COPY statement like this:

```
01  RECORD1 COPY MBR-A.
```

the resultant source would look like this:

```
01  RECORD1.
    05 FIELD-A...
    05 FIELD-B...
```

IBM COBOL does not accept this COPY statement. To compile with IBM COBOL, use the following statement:

```
01  RECORD1.
    COPY MBR-A REPLACING ==01 RECORD-A== BY == ==.
```

CURRENCY-SIGN clause changes—'/', '=', and 'L' characters

OS/VS COBOL with LANGLVL(1), accepted the '/' (slash) character, the 'L' character, and the '=' (equal) sign in the CURRENCY-SIGN clause.

IBM COBOL does not accept these characters as valid. In addition, IBM COBOL does not accept the character G for programs that use DBCS data items.

If these characters are present, you must remove them from the CURRENCY SIGN clause.

Dynamic CALLs to ENTRY points

OS/VS COBOL allowed dynamic CALLs to alternate entry points of subprograms without an intervening CANCEL, in some cases. For information on the situations where the intervening CANCEL is not necessary in OS/VS COBOL, see System Dependencies in Part 6 of *IBM VS COBOL for OS/VS* Release 2.4 (page 408).

IBM COBOL always requires an intervening CANCEL. When converting these programs, add an intervening CANCEL between dynamic CALL statements referencing alternate ENTRY points of subprograms.

EXIT PROGRAM/GOBACK statement changes

In OS/VS COBOL, when an EXIT PROGRAM or GOBACK statement was executed, if the end of range of a PERFORM statement within it had not been reached, the PERFORM statement remaineds in its uncompleted state.

In IBM COBOL, 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 IBM COBOL, status key values have been changed from those received from OS/VS COBOL:

- For QSAM files, see Figure 37 on page 136.
- For VSAM files, see Figure 38 on page 137.

If your OS/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 IBM COBOL file status codes, see the *COBOL Language Reference*.

Figure 37. Status key values—QSAM files

OS/VS	IBM COBOL	Meaning
(undefined)	04	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
00	00	Successful completion
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
90	35	Nonoptional file not present
90	37	Device type conflict
90	39	Conflict of fixed file attributes; OPEN fails
90	96	No file identification (no DD statement for the file)
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
00	48	WRITE attempted when file in OPEN I-O mode
92	49	DELETE or REWRITE attempted when file not in OPEN I-O mode
92	92	Logic error

Figure 38. Status key values—VSAM files

OS/VS	IBM COBOL	Meaning
(unde-fined)	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
00	35	Nonoptional file not present. Can occur when the file is empty.
02	02	Duplicate key, and DUPLICATES specified. Successful completion
10	10	At END (no next logical record). Successful completion
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 IBM COBOL: 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	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	46	Sequential READ attempted with no valid next record
95	39	Conflict of fixed file attributes; OPEN fails
95	95	Invalid or incomplete VSAM file information
96	96	No file identification (no DD statement for this VSAM file)
97	97	OPEN statement execution successful; file integrity verified

IF...OTHERWISE statement changes

OS/VS COBOL allowed IF statements of the nonstandard format:

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

IBM COBOL allows only IF statements having the standard format:

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

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

JUSTIFIED clause changes

Under OS/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 IBM COBOL, 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 IBM COBOL, 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 OS/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 truncated.

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 IBM COBOL, 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.

Numeric class test on group items

OS/VS COBOL allowed the IF NUMERIC class test to be used with group items that contained one or more signed elementary items.

For example, `IF grp1 IS NUMERIC`, when `grp1` is a group item:

```
01 grp1.
   03 yy PIC XX.
   03 mm PIC XX.
   03 dd PIC XX.
```

IBM COBOL issues an S-level message when the IF NUMERIC class test is used for GROUP items whose subordinates are signed.

Numeric data changes

IBM COBOL uses the NUMPROC compiler option to alter the code generated for decimal data. While NUMPROC(MIG) will cause processing very similar to OS/VS COBOL, results are not the same in all cases. The results of MOVEs, comparisons, and arithmetic statements might differ from OS/VS COBOL, particularly when the fields have not been initialized.

Programs that rely on data exceptions to either identify invalid contents of decimal data items or to terminate abnormally might need to be changed to use the class test to validate data in decimal data items.

OCCURS DEPENDING ON clause—ASCENDING/DESCENDING KEY phrase

OS/VS COBOL accepted a variable-length key in the ASCENDING/DESCENDING KEY phrase of the OCCURS DEPENDING ON clauses as an IBM extension.

In IBM COBOL, the ASCENDING/DESCENDING KEY phrase cannot specify a variable-length key.

OCCURS DEPENDING ON clause—value for receiving items changed

In OS/VS COBOL, the current value of the OCCURS DEPENDING ON (ODO) object is always used for both sending and receiving items.

In IBM COBOL, for sending items, the current value of the ODO object is used. For receiving items:

- If a group item contains both the subject and object of an ODO, and is not followed in the same record by a nonsubordinate data item, the maximum length of the item is used.
- If a group item contains both the subject and object of an ODO and is followed in the same record by a nonsubordinate data item, the actual length of the receiving item is used.
- If a group item contains the subject, but not the object of an ODO, the actual length of the 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 location depends on the value of the ODO object, you need to set the object of the OCCURS DEPENDING ON clause before using them in the using phrase of a CALL statement. Under IBM COBOL, for any variable-length group that is not variably located, 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 the second bullet above.

For example:

```
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  VARIABLY-LOCATED-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, IBM COBOL 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 VARIABLY-LOCATED-ITEM follows TABLE-GROUP-2 in the record description. In this case, IBM COBOL 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 phrase—changes in intermediate results

For OS/VS COBOL, the SIZE ERROR phrase for the DIVIDE and MULTIPLY statements applied to both intermediate and final results.

For IBM COBOL, the SIZE ERROR phrase for the DIVIDE and MULTIPLY statements applies only to final results. This is a change between the COBOL 74 Standard and the COBOL 85 Standard. This change might or might not affect your programs.

Therefore, if your OS/VS COBOL program depends upon SIZE ERROR detection for intermediate results, you might need to change it.

Optional word IS

For OS/VS COBOL programs, no MIGR message would be issued if the optional word IS immediately preceded objects within an abbreviated combined relation condition. For example:

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

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

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

Note: IBM COBOL 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
```

PERFORM statement—changes in the VARYING/AFTER phrases

In OS/VS COBOL, in a PERFORM statement with the VARYING/AFTER phrases, 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 augmented by its current BY value.

In IBM COBOL 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 augmented 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 OS/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 IBM COBOL, 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 as in OS/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 OS/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 IBM COBOL, the collating sequence specified in *alphabet-name* is not 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, OS/VS COBOL and IBM COBOL 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 has a PICTURE clause.

RERUN clause changes

When the RERUN clause is specified, OS/VS COBOL takes a checkpoint on the first record; IBM COBOL does not.

RESERVE clause changes

OS/VS COBOL supported the following formats of the FILE CONTROL paragraph RESERVE clause:

```
RESERVE NO ALTERNATE AREA
RESERVE NO ALTERNATE AREAS
RESERVE integer ALTERNATE AREA
RESERVE integer ALTERNATE AREAS
RESERVE integer AREA
RESERVE integer AREAS
```

IBM COBOL supports only the following forms of the RESERVE clause:

```
RESERVE integer AREA
RESERVE integer AREAS
```

If your OS/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 IBM COBOL. That is, the OS/VS COBOL phrase `RESERVE 2 ALTERNATE AREAS` is equivalent to `RESERVE 3 AREASA` in IBM COBOL.

Under OS/VS COBOL with LANGLVL(1), the interpretation of the RESERVE integer AREAS format differed from the interpretation of this format using IBM COBOL.

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 IBM COBOL.

Reserved word list changes

Differences exist between the reserved word list for IBM COBOL and OS/VS COBOL. Appendix B, “COBOL reserved word comparison” on page 190 contains a complete listing of reserved words.

SEARCH statement changes

Under OS/VS COBOL, the ASCENDING/DESCENDING KEY data item could be specified either as the subject or as the object of the WHEN relation-condition of the SEARCH statement.

Under IBM COBOL, 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.

OS/VS COBOL accepted the following. IBM COBOL does not.

```
WHEN VAL = KEY-1 ( INDEX-NAME-1 )
  DISPLAY "TABLE RECORDS OK".
```

The following SEARCH example will execute under both IBM COBOL and OS/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".
```

Segmentation changes—PERFORM statement in independent segments

In OS/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 OS/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.

With IBM COBOL, the compiler does not perform overlay; therefore, the rules given above do not apply.

If your program logic depends upon either of the OS/VS COBOL implementations of these segmentation rules, you must rewrite the program.

SELECT OPTIONAL clause changes

In OS/VS COBOL with LANGLVL(1), if the SELECT OPTIONAL clause is specified in the file control entry, the program will fail if the file is not available.

In IBM COBOL, if the SELECT OPTIONAL clause is specified in the file control entry, the program will **not** fail if the file is not available and a file status code of 05 is returned. A USERMOD can influence this behavior for VSAM. For details, see:

- For MVS: *Language Environment Installation and Customization*.
- For OS/390: *Language Environment for OS/390 Customization*.

SORT special registers

The SORT-CORE-SIZE, SORT-FILE-SIZE, SORT-MESSAGE, and SORT-MODE-SIZE special registers are supported under IBM COBOL, 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 OS/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 IBM COBOL, the SORT-RETURN register only contains codes for successful (RC=0) and unsuccessful (RC=16) SORT completion.

Source language debugging changes

In IBM COBOL and OS/VS COBOL, you can debug source language with the USE FOR DEBUGGING declarative. Valid operands are shown in Figure 39 on page 144. Operands invalid for IBM COBOL must be removed from the OS/VS COBOL program. Use the Debug Tool to achieve the same debugging results.

Figure 39. USE FOR DEBUGGING declarative—valid operands

Debugging operands		Procedures are executed immediately:
OS/VS COBOL	**IBM COBOL**	
procedure-name-1	procedure-name-1	Before each execution of the named procedure. After execution of an ALTER statement referring to the named procedure.
ALL PROCEDURES	ALL PROCEDURES	Before execution of every nondebugging procedure in the outermost program After execution of every ALTER statement in the outermost program (except ALTER statements in declarative procedures).
file-name-n	(none)	See *IBM VS COBOL for OS/VS* for a description.
ALL REFERENCES OF identifier-n	(none)	See *IBM VS COBOL for OS/VS* for a description.
cd-name-1	(none)	See *IBM VS COBOL for OS/VS* for a description.

Subscripts out of range flagged at compile time

IBM COBOL issues an error (RC = 8) 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.

OS/VS COBOL did not issue an equivalent error message.

UNSTRING statements—subscript evaluation changes

In the UNSTRING statements for OS/VS COBOL, any associated subscripting, indexing, or length calculation would be 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 IBM COBOL 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 OS/VS COBOL, subscript IND-1 would be reevaluated before the second receiver ZZ was filled.

Under IBM COBOL, the subscript IND-1 is evaluated only once at the beginning of the execution of the UNSTRING statement.

Under OS/VS COBOL with LANGLVL(1), when the DELIMITED BY ALL phrase 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 phrase in OS/VS COBOL, see *IBM VS COBOL for OS/VS*.

Under IBM COBOL, one or more contiguous occurrences of any delimiters are treated as if they are 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
         ID-R2 DELIMITER IN ID-D2 COUNT IN ID-C2
         ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3
```

OS/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
```

IBM COBOL 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

OS/VS COBOL allowed references to UPSI switches and mnemonic names associated with UPSI. IBM COBOL allows condition-names only.

For example, if a condition-name is defined in the SPECIAL-NAMES paragraph, the following are equivalent:

OS/VS COBOL

```
SPECIAL-NAMES.
    UPSI-0 IS MNUPO

        .
        .
        .
PROCEDURE DIVISION
        .
        .
        .
    IF UPSI-0 = 1 ...
    IF MNUPO = 0 ...
```

IBM COBOL

```
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 ...
```

VALUE clause condition names

For VALUE clause condition names, releases prior to Release 2.4 of OS/VS COBOL allowed the initialization of an alphanumeric field with a numeric value. For example:

```
01 FIELD-A.
    02   LAST-YEAR  PIC XX VALUE 87.
    02   THIS-YEAR  PIC XX VALUE 88.
    02   NEXT-YEAR  PIC XX VALUE 89.
```

IBM COBOL does not accept this language extension. Therefore, to correct the above example, you should code alphanumeric values in the VALUE clauses, as in the following example:

```
01 FIELD-A.
    02   LAST-YEAR  PIC XX VALUE "87".
    02   THIS-YEAR  PIC XX VALUE "88".
    02   NEXT-YEAR  PIC XX VALUE "89".
```

WHEN-COMPILED special register

IBM COBOL and OS/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 **OS/VS COBOL** the format is:

```
hh.mm.ssMMM DD, YYYY  (hour.minute.secondMONTH DAY, YEAR)
```

In **IBM COBOL** the format is:

```
MM/DD/YYhh.mm.ss  (MONTH/DAY/YEARhour.minute.second)
```

WRITE AFTER POSITIONING statement

OS/VS COBOL supported the WRITE statement with the AFTER POSITIONING phrase. IBM COBOL does not.

In IBM COBOL, you can use the WRITE...AFTER ADVANCING statement to receive behavior similar to WRITE...AFTER POSITIONING.

The following two examples show OS/VS COBOL POSITIONING phrases and the equivalent IBM COBOL phrases.

When using WRITE...AFTER ADVANCING with literals:

```
   OS/VS COBOL                                IBM COBOL

AFTER POSITIONING 0                    AFTER ADVANCING PAGE
AFTER POSITIONING 1                    AFTER ADVANCING 1 LINE
AFTER POSITIONING 2                    AFTER ADVANCING 2 LINES
AFTER POSITIONING 3                    AFTER ADVANCING 3 LINES
```

When using WRITE...AFTER ADVANCING with non literals:

```
WRITE OUTPUT-REC AFTER POSITIONING SKIP-CC.
```

```
   OS/VS COBOL                                IBM COBOL
                            SKIP-CC

AFTER POSITIONING SKIP-CC      1         AFTER ADVANCING PAGE
AFTER POSITIONING SKIP-CC     ' '        AFTER ADVANCING 1 LINE
AFTER POSITIONING SKIP-CC      0         AFTER ADVANCING 2 LINES
AFTER POSITIONING SKIP-CC      -         AFTER ADVANCING 3 LINES
```

Note: With IBM COBOL, channel skipping is only supported with references to SPECIAL-NAMES.

CCCA can automatically convert WRITE...AFTER POSITIONING statements. For example, given the following statement:

```
WRITE OUTPUT-REC AFTER POSITIONING n.
```

If n is a literal, CCCA would change the above example 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.

Chapter 11. Compiling converted OS/VS COBOL programs

This chapter describes the differences that exist between the IBM COBOL compilers and the OS/VS COBOL compiler. It contains information on the following topics:

- Key compiler options for converted programs
- Processing compiler options
- Unsupported OS/VS COBOL compiler options
- Prolog format changes

Information specific to any one of these products is noted.

Key compiler options for converted programs

Figure 40 lists the compiler options that have special relevance to converted programs.

Figure 40 (Page 1 of 2). Key compiler options for converted OS/VS COBOL programs

Compiler option	Comments
BUFSIZE	In OS/VS COBOL, the BUF option value specifies the **total number of bytes reserved for buffers.** In IBM COBOL, BUFSIZE specifies the amount of buffer storage reserved for **each compiler work data set**. The default is 4096. If your OS/VS COBOL program uses the BUF option, you must adjust the amount requested in your IBM COBOL BUFSIZE option.
DATA(24)	Use DATA(24) for IBM COBOL programs that are compiled RENT and mixed with OS/VS COBOL programs.
NUMPROC(MIG)	NUMPROC(MIG) processes numeric signs in a way **similar** to, but not exactly like, OS/VS COBOL.
NUMCLS(ALT)	Use NUMCLS(ALT) if you were using the USERMOD shipped with OS/VS COBOL. With the USERMOD, characters A, B, and E (as well as C, D, and F) are considered valid numeric signs in the COBOL Numeric Class Test. (You must also compile with NUMPROC(MIG).) For other alternatives for sign representation, see the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.
OPT(STD)	Use OPT(STD) if you have non referenced data items as eye-catchers or time/version stamps in WORKING-STORAGE. Use OPT(FULL) only if you do not need unused data items.
OUTDD(ddname)	Use this option to override the default ddname (SYSOUT) for SYSOUT output that goes to the system logic output unit. If the ddname is the same as the Language Environment MSGFILE ddname, the output is routed to the ddname designated for MSGFILE. If the ddname is **not** the same as the Language Environment MSGFILE ddname, the output from the DISPLAY statement is directed to the OUTDD ddname destination. If the ddname is not present at first reference, dynamic allocation will take place with the default name and attributes that are specified by Language Environment.
PGMNAME(COMPAT)	Use PGMNAME(COMPAT) to ensure that program names are processed in a manner compatible with OS/VS COBOL.
RMODE(24)	Use RMODE(24) for IBM COBOL programs that are compiled NORENT and mixed with OS/VS COBOL.

Figure 40 (Page 2 of 2). Key compiler options for converted OS/VS COBOL programs

Compiler option	Comments
TRUNC	TRUNC controls the way arithmetic fields are truncated into binary receiving fields during MOVE and arithmetic operations. Use TRUNC(STD) if your shop used TRUNC as the default with OS/VS COBOL. Use TRUNC(OPT) if your shop used NOTRUNC as the default with OS/VS COBOL (except for select program that require guaranteed nontruncation of binary data). For programs that require nontruncation of binary data, use TRUNC(BIN)—especially 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. **Note:** On IBM COBOL, programs compiled with TRUNC(OPT) can give different results than OS/VS COBOL programs compiled with NOTRUNC. Mainly, programs can lose nonzero high-order digits. For statements where loss of high-order digits might take place, IBM COBOL issues a diagnostic message indicating that you should ensure that either: • The sending items will not contain large numbers, or • The receiving items are defined with enough digits in the PICTURE clause to handle the largest sending data items.
WORD(NOOO)	Use WORD(NOOO) if your existing programs use any of the following words and you do not want to change them yet: CLASS-ID, END-INVOKE, INHERITS, INVOKE, LOCAL-STORAGE, METACLASS, METHOD, METHOD-ID, OBJECT, OVERRIDE, RECURSIVE, REPOSITORY, RETURNING, SELF, SUPER. These words are reserved in IBM COBOL for the object-oriented extensions.

Processing compiler options

Your DP shop can use several different methods to control IBM COBOL compilations. The methods are listed below, in their order of precedence:

1. Options “fixed” by your DP shop at installation through the IGYCDOPT module (for MVS) or the IGYCCMSD module (for CMS). Application programmers cannot override these options.
2. Options specified on PROCESS (CBL) statements
3. Options specified with the:
   - MVS JCL PARM parameter
   - CMS COBOL2 command
4. Default options established during installation (options not “fixed” by your DP shop). Unless you override them by specifying other options, your compilation uses these default options.

If you specify a compiler option that conflicts with another, a diagnostic message is issued. The message describes the problem and also describes how the compiler resolved the conflict.

You can specify most of the compiler options on a PROCESS(CBL) statement at the beginning of your program. The exceptions are DUMP, and additionally, on

CMS: DISK, PRINT, and NOPRINT. They are accepted only by the CMS COBOL2 command.

Under IBM COBOL, you do not have to specify a compiler option to process a PROCESS(CBL) statement. (With OS/VS COBOL you must specify the BATCH option to process the CBL statement.)

PARM and PROCESS(CBL) statement options are processed from left to right. For compiler option default values, see the *COBOL Programming Guide*. For information on customizing the default compiler options for your shop, see the:

- For MVS: *Installation and Customization under MVS*
- For OS/390: *Installation and Customization under OS/390*
- For VM: Program Directory

Unsupported OS/VS COBOL compiler options

Figure 41 shows the OS/VS COBOL compiler options that are not supported by IBM COBOL.

For a complete list of compiler options, see Appendix F, “Compiler option comparison” on page 224.

Figure 41 (Page 1 of 2). OS/VS COBOL compiler options not supported by IBM COBOL

OS/VS COBOL option	IBM COBOL equivalent
BATCH/NOBATCH	Batch environment is always available (sequence of programs). **Note:** IBM COBOL considerations for sequence of programs are described in the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.
COUNT/NOCOUNT	Similar function is available in the Debug Tool.
ENDJOB/NOENDJOB	ENDJOB behavior is always in effect.

FLOW/NOFLOW	Similar function is available using the Debug Tool. Compile your program with the TEST(ALL,SYM) option and use the following Debug Tool command:
	`"AT GLOBAL LABEL PERFORM;`
	`LIST LINES %LINE; GO; END-PERFORM;"`

LANGLVL(1/2)	The LANGLVL option is not available. IBM COBOL supports only the COBOL 85 Standard and the COBOL 74 Standard (if using the CMPR2 option) as implemented by VS COBOL II Release 2.
LVL=A|B|C|D/NOLVL	FLAGSTD is used for FIPS flagging. ANSI COBOL 74 FIPS is not supported.
RES/NORES	The RES/NORES option is not available. With IBM COBOL, the object module is always created such that library subroutines are located dynamically at run time, instead of being link-edited with the COBOL program. This is equivalent to RES behavior in OS/VS COBOL.
SUPMAP/NOSUPMAP	Equivalent to the NOCOMPILE/COMPILE compiler option.
SYMDMP/NOSYMDMP	ABEND dumps and dynamic dumps are available through Language Environment services. Symbolic dumps are available through using the TEST compiler option.

Figure 41 (Page 2 of 2). OS/VS COBOL compiler options not supported by IBM COBOL

OS/VS COBOL option	IBM COBOL equivalent
SXREF/NOSXREF	The XREF option provides sorted SXREF output.
VBSUM/NOVBSUM	Function is available with the VBREF compiler option.
CDECK/NOCDECK	The LISTER feature is not supported.
FDECK/NOFDECK	The LISTER feature is not supported.
LCOL1/LCOL2	The LISTER feature is not supported.
LSTONLY/LSTCOMP NOLST	The LISTER feature is not supported.
L120/L132	The LISTER feature is not supported.
OSDECK	With IBM COBOL, the object deck runs in either MVS or VM environments. The OSDECK function is not required.

Prolog format changes

The prolog of an object program is the code that the compiler generates at the entry point of the program. It also contains data that identifies the program.

Object modules generated by IBM COBOL are Language Environment conforming, and thus have a different prolog format than in OS/VS COBOL. You will need to update existing assembler programs that scan for date and time to the new format.

You can compile your programs with the IBM COBOL LIST compiler option to generate a listing that you can use to compare the OS/VS COBOL prolog format with the IBM COBOL prolog format.

Chapter 12. Modifying 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 IBM COBOL with the NOCMPR2 option without change. However, you might need to make modifications based on:

- Three minor COBOL 85 Standard interpretation changes
- New reserved words
- Undocumented VS COBOL II extensions

If your VS COBOL II source programs were compiled with the CMPR2 compiler option, it is recommended that you convert them to NOCMPR2 programs to take advantage of the language enhancements provided by the COBOL 85 Standard, as well as to position your company for future Standard enhancements. For information on language differences between CMPR2 and NOCMPR2 (COBOL 85 Standard) see Appendix L, “Differences between CMPR2 and NOCMPR2” on page 247.

For information on tools that will make the CMPR2 to NOCMPR2 conversion for you automatically, see Appendix C, “Conversion tools for source programs” on page 199.

COBOL 85 Standard interpretation changes

Some language differences exist between programs compiled with NOCMPR2 on VS COBOL II Release 3 (including 3.0, 3.1, and 3.2) and programs compiled with NOCMPR2 on subsequent releases (including VS COBOL II Release 4 and IBM COBOL). 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 for nested programs
- Reference modification of a variable-length group receiver with no length specified

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 IBM COBOL 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.

Precedence of USE procedures

This difference affects the precedence of USE procedures relating to contained programs.

In the VS COBOL II Release 3.x 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 VS COBOL II Release 4 and IBM COBOL, 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 IBM COBOL generated program selects a different USE procedure than would have been used by the VS COBOL II Release 3.x 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.

Reference modification of a variable-length group receiver

Programs that MOVE data to reference-modified, variable-length groups might produce different results depending on whether the length used for the variable-length group is evaluated using the actual length or the maximum length.

You might see a difference if the variable-length group meets all of the following:

- If it is a receiver
- If it contains its own OCCURS DEPENDING ON object
- If it is not followed by a nonsubordinate item (also referred to as a variably-located data item)
- If it is reference-modified and a length is not specified

For example, Group *VAR-LEN-GROUP-A* contains an ODO object and an OCCURS subject and is followed by a variably-located data item.

```
01 VAR-LEN-PARENT-A.
   02 VAR-LEN-GROUP-A.
      03 ODO-OBJECT PIC 99 VALUE 5.
      03 OCCURS-SUBJECT OCCURS 10 TIMES DEPENDING ON ODO-OBJECT.
         04 TAB-ELEM  PIC X(4).
   02 VAR-LOC-ITEM  PIC XX.
01 NEXT-GROUP.

MOVE ALL SPACES TO VAR-LEN-GROUP-A(1:).
```

Group *VAR-LEN-GROUP-B* contains an ODO object and an OCCURS subject and is **not** followed by a variably-located data item. Note, VAR-LOC-ITEM follows the OCCURS subject, but does **not** follow VAR-LEN-GROUP-B.

```
01 VAR-LEN-PARENT-B.
   02 VAR-LEN-GROUP-B.
      03 ODO-OBJECT PIC 99 VALUE 5.
      03 OCCURS-SUBJECT OCCURS 10 TIMES DEPENDING ON ODO-OBJECT.
         04 TAB-ELEM PIC X(4).
      03 VAR-LOC-ITEM PIC XX.
01 NEXT-GROUP.

MOVE ALL SPACES TO VAR-LEN-GROUP-B(1:).
```

In the above examples, `MOVE ALL SPACES TO VAR-LEN-GROUP-A (1:)` would give the same results with any NOCMPR2 program (VS COBOL II Release 3.x, VS COBOL II Release 4, or IBM COBOL). They all use the actual length in this case.

Whereas, `MOVE ALL SPACES TO VAR-LEN-GROUP-B (1:)` would give different results for the following programs compiled with NOCMPR2:

- VS COBOL II Release 3.x uses the actual length of the group as defined by the current value of the ODO object (the actual length of the group is set to spaces using the ODO object value).
- VS COBOL II Release 4 and IBM COBOL use the maximum length of the group (the entire data item is set to spaces using the ODO object value).

If a program contains a reference-modified, variable-length group receiver that: contains its own ODO object; is not followed by variably-located data; 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.

ACCEPT statement

One additional difference between later releases and VS COBOL II Release 3.0 only involves the system input devices for the mnemonic-name suboption of the ACCEPT statement.

For Release 3.0 only, an input record of 80 characters is assumed even if a logical record length of other than 80 characters is specified.

In IBM COBOL, the maximum logical record length allowed is 256 characters.

New reserved words

Both COBOL for MVS & VM and COBOL/370 introduced new reserved words, which are listed on “Reserved words” on page 268.

COBOL for MVS & VM and COBOL for OS/390 & VM provide an alternate reserved word table, which does not reserve the new words added for the object-oriented extensions. Specify the WORD(NOOO) compiler option, to continue using any of the words above, except FUNCTION and PROCEDURE-POINTER.

You must remove FUNCTION and PROCEDURE-POINTER if they are used in existing programs.

Undocumented VS COBOL II extensions

VS COBOL II allows you to code a period in Area A following an invalid Area A item (or no item). With IBM COBOL periods in Area A must be preceded by a valid Area A item.

Chapter 13. Compiling VS COBOL II programs

This chapter describes the differences that exist between the IBM COBOL compilers and the VS COBOL II compiler. This chapter contains information on the following topics:

- Key compiler options for VS COBOL II programs
- Prolog format changes

Key compiler options for VS COBOL II programs

The IBM COBOL and VS COBOL II compilers are very similar. If you will be using 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 converting from CMPR2 to NOCMPR2, see Appendix L, “Differences between CMPR2 and NOCMPR2” on page 247. For information on other compiler options, see the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.

When compiling with IBM COBOL

Figure 42 lists the IBM COBOL compiler options that have special relevance to converted programs.

Figure 42. Key IBM COBOL compiler options for converted VS COBOL II programs

IBM COBOL compiler option	Comments
PGMNAME(COMPAT)	If compiling with IBM COBOL, use this option to ensure that program names are processed in a manner compatible with VS COBOL II (and COBOL/370).
RMODE(AUTO)	Use RMODE(AUTO) or RMODE(24) for IBM COBOL NORENT programs that pass data to programs running in AMODE(24).
TEST	The syntax of the TEST option is different in IBM COBOL than in VS COBOL II. 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 the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.
WORD(NOOO)	Use WORD(NOOO) if your existing programs use any of the following words, which are now reserved in IBM COBOL for the object-oriented COBOL extensions: CLASS-ID, END-INVOKE, INHERITS, INVOKE, LOCAL-STORAGE, METACLASS, METHOD, METHOD-ID, OBJECT, OVERRIDE, RECURSIVE, REPOSITORY, RETURNING, SELF, SUPER.

Figure 43 list the VS COBOL II compiler options that are not available in IBM COBOL. In some cases, the function of the VS COBOL II compiler option is mapped to an IBM COBOL compiler option, as described in the comments section.

Figure 43. Compiler options not available in IBM COBOL

VS COBOL II compiler option	Comments
FDUMP/NOFDUMP	IBM COBOL does not provide the FDUMP compiler option. For existing applications, FDUMP is mapped to the IBM COBOL TEST(SYM) compiler option, which can provide equivalent function and more. Language Environment generates a better formatted dump than VS COBOL II, regardless of the FDUMP option. But, the presence of FDUMP enables Language Environment to include the symbolic dump of information about data items in the formatted dump. For information on how to obtain the Language Environment formatted dump at abnormal termination, see *Language Environment Debugging Guide and Run-Time Messages*. If NOFDUMP is encountered, IBM COBOL issues a warning message because NOFDUMP is not supported.
FLAGSAA	IBM COBOL does not support the FLAGSAA option. If FLAGSAA is specified, IBM COBOL issues a W-level message.
RES/NORES	IBM COBOL does not provide the RES/NORES compiler option. If RES is encountered, an informational message is issued. If NORES is encountered, a warning message is issued.

Prolog format changes

The prolog of an object program is the code that the compiler generates at the entry point of the program. It also contains data that identifies the program.

Object modules generated by IBM COBOL are Language Environment 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 IBM COBOL LIST compiler option to generate a listing that you can use to compare the VS COBOL II format with the IBM COBOL format.

Chapter 14. Upgrading compiler release levels

When upgrading your source programs from COBOL/370 or COBOL for MVS & VM to a later release, you have the following considerations:

- Compiler option considerations
- Determining which programs require upgrade

Compiler option considerations

The compiler options in COBOL for OS/390 & VM and COBOL for MVS & VM are slightly different than the compiler options in COBOL/370. Figure 44 lists the differences.

Figure 44 (Page 1 of 2). Compiler options differences

Compiler option	Comments
ARITH(COMPAT)	Same as VS COBOL II to COBOL for OS/390 & VM Version 2 Release 1. For intermediate results in arithmetic statements.
EVENTS	The EVENTS option is not available. To emulate the COBOL/370 EVENTS compiler option: 1. Specify the ADATA compiler option. 2. Allocate SYSADATA and SYSEVENTS. 3. Use the ADEXIT suboption of the EXIT compiler option with the sample exit program IGYADXIT.
INTDATE	Use INTDATE(ANSI) to get the same results as COBOL/370 Release 1 for date intrinsic functions. Use INTDATE(LILIAN) if you store integer values and will be using other languages with the same data. INTDATE(LILIAN) will cause the date Intrinsic Functions to use the Language Environment start date, which is the same starting date that would be used by PL/I or C programs that use Language Environment date callable services. If integer dates are used only within a single program, such as converting Gregorian to Lilian and back to Gregorian in the same program, the setting of INTDATE is immaterial.
PGMNAME	Use PGMNAME(COMPAT) to ensure program names are processed in a similar manner as OS/VS COBOL, VS COBOL II and COBOL/370.

Figure 44 (Page 2 of 2). Compiler options differences

Compiler option	Comments
TRUNC(BIN)	In releases of COBOL for OS/390 & VM prior to Version 2 Release 2, unsigned binary data items with TRUNC(BIN) were correctly supported only when the binary value contained at most 15 bits for halfwords, 31 bits for fullwords, or 63 bits for doublewords. In other words, the sign bit was not used as part of the numeric value when the data item was unsigned. With COBOL for OS/390 & VM Version 2 Release 2, all 16 bits of a halfword, all 32 bits of a fullword, and all 64 bits of a doubleword can be used as part of the numeric value of an unsigned COMP-5 data item or an unsigned binary data item with TRUNC(BIN). For example, in a program compiled with TRUNC(BIN), a data item declared like this `01 X pic 9(2) binary.` correctly supported binary values from 0 through 32767 in prior releases, but with Version 2 Release 2 now supports values of 0 through 65535. This support necessarily yields different arithmetic results than were obtained with the prior releases, in the case that these very large unsigned binary values were inadvertently used.
WORD(NOOO)	Use WORD(NOOO) if your existing programs use any of the following words, which are now reserved in COBOL for MVS & VM for the object-oriented extensions: CLASS-ID, END-INVOKE, INHERITS, INVOKE, LOCAL-STORAGE, METACLASS, METHOD, METHOD-ID, OBJECT, OVERRIDE, RECURSIVE, REPOSITORY, RETURNING, SELF, SUPER.

Determining which programs require upgrade

When using OS/390 Release 3 or later you must make changes in your JCL for building object-oriented COBOL applications.

With COBOL for MVS & VM, the SOM/MVS kernel and the SOM/MVS class libraries were linked together with the COBOL application load module. With COBOL for OS/390 & VM, running under OS/390 Release 3 or later, the SOM/MVS kernel and class libraries must be accessed from the SOM/MVS DLLs. You need to make the following changes:

- Change your JCL to specify the COBOL for OS/390 & VM DLL compiler option.
- Change your JCL so that the prelink step specifies (at minimum) the DLL definition sidedecks for the SOM kernel and class libraries.
- In the prelink step, eliminate the SOM SGOSPLKD data set from the SYSLIB concatenation.

Additionally, the format of the SOM profile data set has changed and you specify the profile using the SOMENV DD statement rather than the SOMPROF DD statement as in COBOL for MVS & VM.

Chapter 15. CICS conversion considerations—source

This chapter explains the source language considerations for programs running on CICS. It describes the action you need to take for applications, which use source language (either CICS source or OS/VS COBOL source), that involve the following function:

- Compiler options relevant for programs run on CICS
- Base addressability considerations

CICS run-time considerations are included in the following chapters:

- Chapter 6, “Moving from the OS/VS COBOL run time” on page 58
- Chapter 7, “Moving from the VS COBOL II run time” on page 70

The CICS environment supports the use of IBM COBOL and Language Environment beginning with CICS Version 3 Release 3 or later. You can compile and execute CICS command-level application programs. (CICS macro-level programs will neither compile with IBM COBOL, nor execute with Language Environment. The CICS Application Migration Aid can help with this conversion. For details, see “CICS Application Migration Aid” on page 206.)

Compiler options relevant for programs run on CICS

Figure 45 lists compiler options relevant to IBM COBOL programs running on CICS.

Figure 45 (Page 1 of 2). Compiler options relevant for programs run on CICS

Compiler option	Comments
DATA(24)	Use DATA(24) in the following two cases: • For batch programs that use the CICS shared database support (DFHDRP), any parameters passed to CBLTDLI must be below the 16M line. • For CICS on-line programs that access a local DL/I database using CALL CBLTDLI, the parameter list and all storage areas that are referred to in the parameter list of the DL/I call must reside below the 16M line. For CICS/ESA you must specify TASKDATALOC(BELOW) in the TRANSACTION definition for any transaction that involves CICS online programs that access a local DL/I database using CALL CBLTDLI. **Note:** DATA(24) is not required when you are using DBCTL. For details, see the *CICS-IMS Database Control Guide for CICS/ESA*.
NODYNAM	NODYNAM is required for programs translated by the CICS translator. The NODYNAM compiler option is required in this case because the CICS command level stub cannot be dynamically called.
RENT	RENT is required for CICS programs. RENT causes the compiler to produce reentrant code and allows you to place the COBOL modules in the MVS Link Pack Area or Extended Link Pack Area and thus shared among multiple address spaces under CICS. Also, the modules cannot be overwritten, since the LPA/ELPA have a storage protect key.

Figure 45 (Page 2 of 2). Compiler options relevant for programs run on CICS

Compiler option	Comments
TRUNC(OPT) TRUNC(BIN)	Use TRUNC(OPT) for CICS programs containing EXEC CICS commands, if the program's usage of binary data items conforms to the PICTURE and USAGE clause for those data items. Use TRUNC(BIN) if your program's usage 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.

Base addressability considerations

Using OS/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, an OS/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 IBM COBOL, and the associated support within CICS, this is no longer necessary. Therefore, when you upgrade from OS/VS COBOL to IBM COBOL, you must convert such programs. (If the COBOL/CICS program does not manipulate BLL cells addressing, conversion is not necessary.)

Note: You can do this automatically using the CCCA, as explained in Appendix C, “Conversion tools for source programs” on page 199.

The three areas that require change for programs that manipulate BLL cells are:

- SERVICE RELOAD statement changes
- LENGTH OF special register
- Programs using BLL cells

SERVICE RELOAD statement changes

In OS/VS COBOL, for programs to be executed under CICS, the SERVICE RELOAD statement is required to ensure addressability of items defined in the LINKAGE SECTION.

With IBM COBOL, the SERVICE RELOAD statement is not required. If encountered it is treated as a comment.

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.

Programs using BLL cells

The following steps summarize the conversion of an OS/VS COBOL CICS program to an IBM COBOL CICS program. For more information, see *CICS/ESA Application Programming Guide*.

1. Remove all SERVICE RELOAD statements. The IBM COBOL compilers treat 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 OS/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:*** You can convert your programs to avoid the usage of BLL
| cells for DL/I calls as well. 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.

| The following example shows how to get the address of the PCBs into
| PCB1-ADDR and PCB2-ADDR.

```
 LINKAGE SECTION.
     COPY DLIUIB.
 01  OVERLAY-DLIUIB REDEFINES DLIUIB.
     02  PCBADDR USAGE IS POINTER.
     02  FILLER  PIC XX.

 01  PCB-ADDRESSES.
     02  PCB1-ADDR    USAGE IS POINTER.
     02  PCB2-ADDR    USAGE IS POINTER.

* VACATION PCB
 01  VAC-PCB.
 COPY RDLICVP.

* HOTEL PCB
 01 HTL-PCB.
 COPY RDLICHT.
⋮
 PROCEDURE DIVISION.
⋮
     CALL 'CBLTDLI' USING PCB
                          PSBNAME
                          ADDRESS OF DLIUIB

     IF UIBFCTR IS NOT EQUAL TO LOW-VALUES
         MOVE 'PCB CALL FAILED.' TO MSG-TEXT
         PERFORM WRITE-MESSAGE-AND-AMEND
     ELSE
         SET ADDRESS OF PCB-ADDRESSES TO PCBADDR
         SET ADDRESS OF HTL-PCB TO PCB1-ADDR
         SET ADDRESS OF VAC-PCB TO PCB2-ADDR
     END-IF
⋮
```

Note: In the code above note that lines "SET ADDRESS OF PCB-ADDRESSES TO PCBADDR" through "SET ADDRESS OF VAC-PCB TO PCB2-ADDR" are the commands that obtain the adresses of the required PCB from the address list in the UIB.

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; with IBM COBOL, BLL cells are no longer explicitly defined in the LINKAGE SECTION.
2. In 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.

OS/VS COBOL

```
LINKAGE SECTION.
 01  PARAMETER-LIST.
    05  PARM-FILLER          PIC S9(8) COMP.
    05  PARM-AREA1-POINTER   PIC S9(8) COMP.
    05  PARM-AREA2-POINTER   PIC S9(8) COMP.
 01  AREA1.
    05  AREA1-DATA           PIC X(100).
 01  AREA2.
    05  AREA2-DATA           PIC X(100).
          .
          .
  PROCEDURE DIVISION.
          .
          .
    EXEC CICS READ DATASET("INFILE")
       RIDFLD(INFILE-KEY)
       SET(PARM-AREA1-POINTER)
       LENGTH(RECORD-LEN)
    SERVICE RELOAD PARM-AREA1-POINTER.
```

IBM COBOL

```
LINKAGE SECTION.


01  AREA1.
   05  AREA1-DATA     PIC X(100).
01  AREA2.
   05  AREA2-DATA     PIC X(100).
 .
 .
PROCEDURE DIVISION.
 .
 .
EXEC CICS READ DATASET("INFILE")
    RIDFLD(INFILE-KEY)
    SET(ADDRESS OF AREA1)
    LENGTH(RECORD-LEN).
```

Example 2: Processing storage areas exceeding 4K

In OS/VS COBOL, if a LINKAGE SECTION area was greater than 4096 bytes in length, then you would have had to include statements to provide addressability to the entire storage area. For IBM COBOL programs this is no longer necessary.

The following example shows the coding for both an OS/VS COBOL program and an IBM COBOL 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.

OS/VS COBOL

```
LINKAGE SECTION.
 01 PARMLIST.
    .
    .
    05  RECORD-POINTERA  PIC S9(8) COMP.
    05  RECORD-POINTERB  PIC S9(8) COMP.
    .
    .
 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.
```

IBM COBOL

```
LINKAGE SECTION.

01 FILE-RECORD.
   05  REC-DATA1       PIC X(2500).
   05  REC-DATA2       PIC X(2500).
       .
       .
PROCEDURE DIVISION.
       .
       .
    EXEC CICS READ DATASET("INFILE")
         RIDFLD(INFILE-KEY)
         SET(ADDRESS OF FILE-RECORD)
         LENGTH(RECORD-LEN)
    END-EXEC
```

Example 3: Accessing chained storage areas

In an OS/VS COBOL CICS program you would access chained storage areas by defining a storage area in the LINKAGE SECTION that contained a pointer to another storage area. You would then access the next chained area by copying the address of the next area into the associated BLL. It was also necessary to code a paragraph name following any statement that modified the contents of the BLL cell used to address the chained areas. With IBM COBOL this is simplified by using the SET statement to set the address of the chained area.

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 IBM COBOL 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.)

OS/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).
    .
    .
PROCEDURE DIVISION.
    .
    .
    MOVE CHS-NEXT-AREA TO CHAINED-POINTER.
ANY-PARAGRAPH-NAME.
    MOVE CHS-DATA TO WSDATA-HOLD.
    .
    .
```

IBM COBOL

```
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.
      MOVE CHS-DATA TO WS-DATA-HOLD.
      .
      .
      .
```

Example 4: Using the OCCURS DEPENDING ON clause

In OS/VS COBOL, if the LINKAGE SECTION contained the object of the OCCURS DEPENDING ON clause, when the number of occurences of the subject of the OCCURs clause changed you would need to reset the object of the OCCURS DEPENDING ON clause to trigger the update of the group length. In IBM COBOL this is unecessary, the length of the group is calculated whenever the structure is referenced.

During conversion, in your IBM COBOL program, remove any code that resets the contents of the object of the OCCURS DEPENDING ON clause. (These references are no longer necessary.)

OS/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 VAR-RECORD TO WS-RECORD-HOLD.
     .
     .
```

IBM COBOL

```
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.
     .
     .
```

Adding IBM COBOL programs to existing COBOL applications

Chapter 16. Adding IBM COBOL programs

When you add an IBM COBOL program to an existing application, you are either recompiling an existing program with IBM COBOL or including a newly written IBM COBOL program. When you add IBM COBOL programs to your existing applications you have the ability to:

- Use a four-digit year in the date function for new applications
- Interpret two-digit year data as four-digit years for existing applications
- Use Language Environment condition handling
- Upgrade your existing programs incrementally, as your shop's needs dictate

Restriction: On CICS, you cannot mix OS/VS COBOL programs and IBM COBOL programs in the same run unit. (EXEC CICS LINK and EXEC CICS XCTL will create a separate run unit.)

Important!

Once you add an IBM COBOL program to an existing application, that application must run under Language Environment.

This chapter includes information on the following:

- Applications comprised of RES programs
- Applications comprised of NORES programs
- Multiple load module considerations
- AMODE and RMODE considerations
- Run-time considerations

Once you begin adding IBM COBOL programs to your existing applications, you need to know the implications of link-editing existing applications with Language Environment. First, you must use the SCEELKED link-edit library. When SCEELKED is used when you link-edit, it impacts the remaining programs in the applications. How it effects existing applications depends on whether the application is comprised of:

- Programs compiled RES
- Programs compiled NORES
- Multiple load modules

Applications comprised of RES programs

When you add an IBM COBOL program to an application comprised of programs compiled RES, you need to:

- Link-edit the IBM COBOL program with Language Environment
- Understand the requirements for using dynamic and static CALLs

The following sections list the requirements for using CALLs on non-CICS and on CICS.

Adding IBM COBOL programs that do static CALLs

Under both CICS and non-CICS, if you create a COBOL for OS/390 & VM or COBOL for MVS & VM program that makes static CALLs to a VS COBOL II program, when you link-edit the load module with Language Environment, note:

- The link-edit job must contain a REPLACE statement to replace IGZEBST (the VS COBOL II bootstrap) for VS COBOL II programs compiled with the RES compiler option that were link-edited with the VS COBOL II Release 4 run time without APAR PN74000 applied.
- The link-edit job must contain a REPLACE statement to replace IGZEBST for VS COBOL II programs compiled with the RES compiler option that were link-edited with Language Environment Release 2 or Release 3 without APAR PN74011 applied.

If you do not have the correct level of the IGZEBST routine, you will encounter a program check when the VS COBOL II program is called.

To have the best CALL performance, it is recommended that the link-edit job contain a REPLACE statement for the following:

- For COBOL/370 programs— the IGZCBSN bootstrap if the COBOL/370 programs were link-edited with Language Environment Release 2, Release 3, or Release 4 without APAR PN74011 applied.
- For VS COBOL II programs compiled with the RES compiler option, the IGZEBST bootstrap if the VS COBOL II RES programs were link-edited with Language Environment Release 4 without APAR PN74011 applied.

For an example of link-edit JCL or sample jobs in Language Environment library SCEESAMP, members IGZWRLKA, IGZWRLKB, and IGZWRLKC see Appendix K, “Link-edit example” on page 245.

CALLs on non-CICS

For CALLs between OS/VS COBOL programs and IBM COBOL programs, parameters must be below the 16M line. The following sections explain the action you need to take for dynamic and static CALLs.

Dynamic CALLs

Load modules that contain an IBM COBOL program that dynamically CALLs an OS/VS COBOL program must be addressable by the OS/VS COBOL program. Specifying the appropriate IBM COBOL compiler option, depending on whether the IBM COBOL program is compiled with RENT or NORENT ensures that the data is addressable by the OS/VS COBOL program.

For IBM COBOL programs compiled RENT, specify the DATA(24) compiler option.

For IBM COBOL programs compiled NORENT, specify the RMODE(24) or RMODE(AUTO) compiler option.

Static CALLs

If you issue static CALLs with OS/VS COBOL and IBM COBOL programs, thus forming a single load module, the load module must reside below the 16M line. The load module must be marked RMODE(24), AMODE(24).

For load modules with both IBM COBOL and OS/VS COBOL programs, you must override the default AMODE setting to AMODE(24) when the load module contains an IBM COBOL program compiled with NORENT. (For programs compiled with RENT, no action is necessary. The linkage editor automatically assigns the correct AMODE setting.) For instructions on how to override the default AMODE setting, see Appendix J, “Overriding linkage editor defaults” on page 244.

CALLs on CICS

With IBM COBOL, you can use static and dynamic CALLs to VS COBOL II, IBM COBOL, assembler, C, and PL/I programs. However, Language Environment does not support static or dynamic CALLs between IBM COBOL programs and OS/VS COBOL programs, (Continue to access OS/VS COBOL subroutines by EXEC CICS LINK.)

General considerations

If your IBM COBOL program was processed by the CICS translator, a CALLing program must pass to the CALLed IBM COBOL program the CICS EXEC interface block (DFHEIBLK) and the communication area (DFHCOMMAREA) as the first two parameters of the CALL statement. If the IBM COBOL 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.

Static CALLs

With IBM COBOL, you can use the COBOL CALL statement to statically CALL VS COBOL II, IBM COBOL, and assembler programs. For details of when static CALLs are supported, see Figure 53 on page 212. In addition, Language Environment supports ILC between PL/I and COBOL and between C and COBOL when running on CICS. For details, see the *Language Environment Writing Interlanguage Applications*.

With OS/VS COBOL, if multiple COBOL programs are separately compiled and then link-edited together, only the first program can contain CICS requests. With IBM COBOL, this restriction is removed, giving you greater flexibility in application program design.

Dynamic CALLs

With IBM COBOL, you can use the COBOL CALL statement to dynamically call VS COBOL II, IBM COBOL, and assembler programs. For details of when dynamic CALLs are supported, see Figure 53 on page 212. For example, dynamically CALLed programs can contain CICS requests. In addition, Language Environment supports ILC between PL/I and COBOL and between C and COBOL when running on CICS. For details, see the *Language Environment Writing Interlanguage Applications*.

For IBM COBOL programs running under CICS, the ON EXCEPTION/OVERFLOW clause of the CALL statement is enabled for programs compiled with the NOCMPR2 compiler option. Additionally, dynamically CALLed programs can contain CICS requests.

Applications comprised of NORES programs

When you add an IBM COBOL program to a load module comprised of programs compiled NORES, you need to:

- Link-edit the IBM COBOL program with Language Environment
- Link-edit all other programs in the application with Language Environment and understand how the behavior of these NORES programs changes when link-edited with Language Environment
- Link-edit REPLACE all the IGZ and ILBO CSECTs with the copies from Language Environment. For an example of link-edit JCL that shows how to replace the current library routines in a load module with the Language Environment library routines, see Appendix K, “Link-edit example” on page 245.

Behavior before link-edit with Language Environment

Before adding the IBM COBOL program, link-editing an application with all NORES programs with Language Environment was not required.

If you look back to:

- Chapter 6, “Moving from the OS/VS COBOL run time” on page 58
- Chapter 7, “Moving from the VS COBOL II run time” on page 70

you will notice that programs compiled NORES that are not link-edited with Language Environment are not affected by many of the subjects in those two chapters. This is because programs compiled NORES do not access the Language Environment library, but continue to run in the environment in which they were link-edited.

Behavior after link-edit with Language Environment

After adding the IBM COBOL program and link-editing the remaining programs with Language Environment, NORES programs are now “RES-like.” Many of the considerations for programs compiled RES now apply to these NORES programs that you have link-edited with Language Environment. For details, see “Implications of becoming RES-like” on page 100.

Link-edit override requirement

For load modules with both IBM COBOL and OS/VS COBOL programs, you must override the default AMODE setting to AMODE(24) when the load module contains an IBM COBOL program compiled with NORENT. For instructions on how to override the default AMODE setting, see Appendix J, “Overriding linkage editor defaults” on page 244.

Multiple load module considerations

Applications with multiple load modules might not be supported under Language Environment. To determine whether a multiple load module application is supported, you need to know:

- The main module
- The main program of the main module
- The sub module(s)

Note: When you link edit an OS/VS COBOL NORES or VS COBOL II NORES program that is part of a multiprogram load module with IBM COBOL, the COBOL

library routines in the load module must be replaced with the Language Environment library routines. Failure to do so can cause unpredictable results. For a coding example of how to replace the library routines, see Appendix K, "Link-edit example" on page 245.

OS/VS COBOL considerations

Figure 46 lists all the possible combinations of multiple load modules for OS/VS COBOL programs. In the following table, if the load module has multiple programs, the main program is listed first.

Figure 46 (Page 1 of 2). Support for applications with multiple load modules—OS/VS COBOL

Main module	Sub module	Support for	Link-edit with LanEnv required
OS/VS COBOL NORES[1]	IBM COBOL only	No	n/a
	OS/VS COBOL RES only	No	n/a
	OS/VS COBOL NORES only	Yes	No
	IBM COBOL & OS/VS COBOL RES	No	n/a
	IBM COBOL & OS/VS COBOL NORES	No	n/a
	OS/VS COBOL RES & OS/VS COBOL NORES	No	n/a
	IBM COBOL & OS/VS COBOL RES & OS/VS COBOL NORES	No	n/a
OS/VS COBOL RES	All combinations	Yes	Yes[2]
IBM COBOL	All combinations	Yes	Yes[2]
IBM COBOL & OS/VS COBOL RES	All combinations	Yes	Yes[2]
OS/VS COBOL RES & IBM COBOL	All combinations	Yes	Yes[2]
IBM COBOL & OS/VS COBOL NORES	All combinations	Yes	Yes[2]
OS/VS COBOL NORES & IBM COBOL	All combinations	Yes	Yes[2]
OS/VS COBOL RES & OS/VS COBOL NORES	n/a	No[3]	n/a
OS/VS COBOL NORES & OS/VS COBOL RES	n/a	No[3]	n/a
IBM COBOL & OS/VS COBOL RES & OS/VS COBOL NORES	All combinations	Yes	Yes[2]
OS/VS COBOL RES & OS/VS COBOL NORES & IBM COBOL	All combinations	Yes	Yes[2]

Figure 46 (Page 2 of 2). Support for applications with multiple load modules—OS/VS COBOL

Main module	Sub module	Support for	Link-edit with LanEnv required
OS/VS COBOL NORES & OS/VS COBOL RES & IBM COBOL	All combinations	Yes	Yes[2]

Note:

1 Load module that contains only OS/VS COBOL NORES programs can access submodules only if it uses an assembler program to load or link to the submodule.

2 Link-editing with Language Environment is not required when the sub module contains only OS/VS COBOL RES programs.

3 Load modules that contain OS/VS COBOL programs compiled RES and OS/VS COBOL programs compiled NORES are not supported unless you include an IBM COBOL program or certain CSECTs. For details, see "Applications with COBOL programs compiled RES and NORES" on page 59.

IBM COBOL represents a COBOL/370, COBOL for MVS & VM, or COBOL for OS/390 & VM program.

All Combinations is synonymous to the seven different combinations of programs that the sub module can consist of (as listed next to OS/VS COBOL NORES at the beginning of the table).

VS COBOL II considerations

Figure 47 lists possible combinations of multiple load modules for VS COBOL II programs. In the following table, if the load module has multiple programs, the main program is listed first.

.

Figure 47 (Page 1 of 2). Support for applications with multiple load modules—VS COBOL II

Main module	Sub module	Support for	Link-edit with LanEnv required
VS COBOL II NORES	IBM COBOL only	Yes	Yes
	VS COBOL II RES only	Yes	Yes
	VS COBOL II NORES only	Yes	No
	IBM COBOL & VS COBOL II RES	Yes	Yes
	IBM COBOL & VS COBOL II NORES	Yes	Yes
	VS COBOL II RES & VS COBOL II NORES	Yes	Yes
	IBM COBOL & VS COBOL II RES & VS COBOL II NORES	Yes	Yes
VS COBOL II RES	IBM COBOL only	Yes	No
	VS COBOL II RES only	Yes	No
	VS COBOL II NORES only	Yes	Yes
	IBM COBOL & VS COBOL II RES	Yes	No
	IBM COBOL & VS COBOL II NORES	Yes	Yes
	VS COBOL II RES & VS COBOL II NORES	Yes	Yes
	IBM COBOL & VS COBOL II RES & VS COBOL II NORES	Yes	Yes
IBM COBOL	IBM COBOL only	Yes	No
	VS COBOL II RES only	Yes	No
	VS COBOL II NORES only	Yes	Yes
	IBM COBOL & VS COBOL II RES	Yes	No
	IBM COBOL & VS COBOL II NORES	Yes	Yes
	VS COBOL II RES & VS COBOL II NORES	Yes	Yes
	IBM COBOL & VS COBOL II RES & VS COBOL II NORES	Yes	Yes

Figure 47 (Page 2 of 2). Support for applications with multiple load modules—VS COBOL II

Main module	Sub module	Support for	Link-edit with LanEnv required
IBM COBOL & VS COBOL II RES	IBM COBOL only	Yes	No
	VS COBOL II RES only	Yes	No
	VS COBOL II NORES only	Yes	Yes
	IBM COBOL & VS COBOL II RES	Yes	No
	IBM COBOL & VS COBOL II NORES	Yes	Yes
	VS COBOL II RES & VS COBOL II NORES	Yes	Yes
	IBM COBOL & VS COBOL II RES & VS COBOL II NORES	Yes	Yes
VS COBOL II RES & IBM COBOL	IBM COBOL only	Yes	No
	VS COBOL II RES only	Yes	No
	VS COBOL II NORES only	Yes	Yes
	IBM COBOL & VS COBOL II RES	Yes	No
	IBM COBOL & VS COBOL II NORES	Yes	Yes
	VS COBOL II RES & VS COBOL II NORES	Yes	Yes
	IBM COBOL & VS COBOL II RES & VS COBOL II NORES	Yes	Yes
IBM COBOL & VS COBOL II NORES	All combinations	Yes	Yes
VS COBOL II NORES & IBM COBOL	All combinations	Yes	Yes
VS COBOL II RES & VS COBOL II NORES	All combinations	Yes	Yes
VS COBOL II NORES & VS COBOL II RES	All combinations	Yes	Yes
IBM COBOL & VS COBOL II RES & VS COBOL II NORES	All combinations	Yes	Yes
VS COBOL II RES & VS COBOL II NORES & IBM COBOL	All combinations	Yes	Yes
VS COBOL II NORES & VS COBOL II RES & IBM COBOL	All combinations	Yes	Yes

Note:

In general, any multiple load module application that contains a VS COBOL II NORES program must be link-edited with Language Environment.

IBM COBOL represents a COBOL/370, COBOL for MVS & VM, or COBOL for OS/390 & VM program.

All Combinations is synonymous to the seven different combinations of programs that the sub module can consist of (as listed next to VS COBOL II NORES at the beginning of the table).

For additional link edit requirements, see “Adding IBM COBOL programs that do static CALLs” on page 171.

AMODE and RMODE considerations

All OS/VS COBOL programs are AMODE(24) and RMODE(24). IBM COBOL programs are always AMODE(ANY) and can be either RMODE(24) or RMODE(ANY). The WORKING-STORAGE data items can be either above or below the line, based on the DATA, RENT and RMODE compiler option..

OS/VS COBOL programs can CALL IBM COBOL programs without AMODE problems, since both programs can access below-the-line data. When an IBM COBOL

program CALLs an OS/VS COBOL program, you can get data exceptions if your data or parameter list is above the line.

To avoid data exceptions, ensure that all of your data and parameter lists are below the line by compiling with DATA(24) for RENT programs, or RMODE(24) or RMODE(AUTO) for NORENT programs.

Figure 48 shows the results of different combinations of compiler options and compilers. All CALLs are dynamic and represented by arrows. The solid lines represent valid CALLs; the dotted line represents an invalid CALL.

P1 and P5 are OS/VS COBOL programs, so the WORKING-STORAGE data items are included in the object module, and must be below the 16M line. For IBM COBOL programs compiled with the RENT option, WORKING-STORAGE data items are separate from the object module, and their location is controlled by the DATA compiler option (as with programs P2 and P3). For programs compiled with NORENT, the WORKING-STORAGE data items are included in the object module, so their location depends on the RMODE option (as with program P4).

All of these CALLs will work successfully, except for P3 CALLing P5. Because P3 was compiled with RENT and DATA(31), its WORKING-STORAGE and any parameter lists are located above the 16M line. This means that even if P3 is passing parameters that it received from P2, the parameter list cannot be addressed by P5, so the CALL will fail. The CALL will also fail if the parameters themselves are above the 16M line, such as data items in the WORKING-STORAGE SECTION of P1 and P3.

Note: Static CALLs from AMODE=31 programs to OS/VS COBOL programs will always fail.

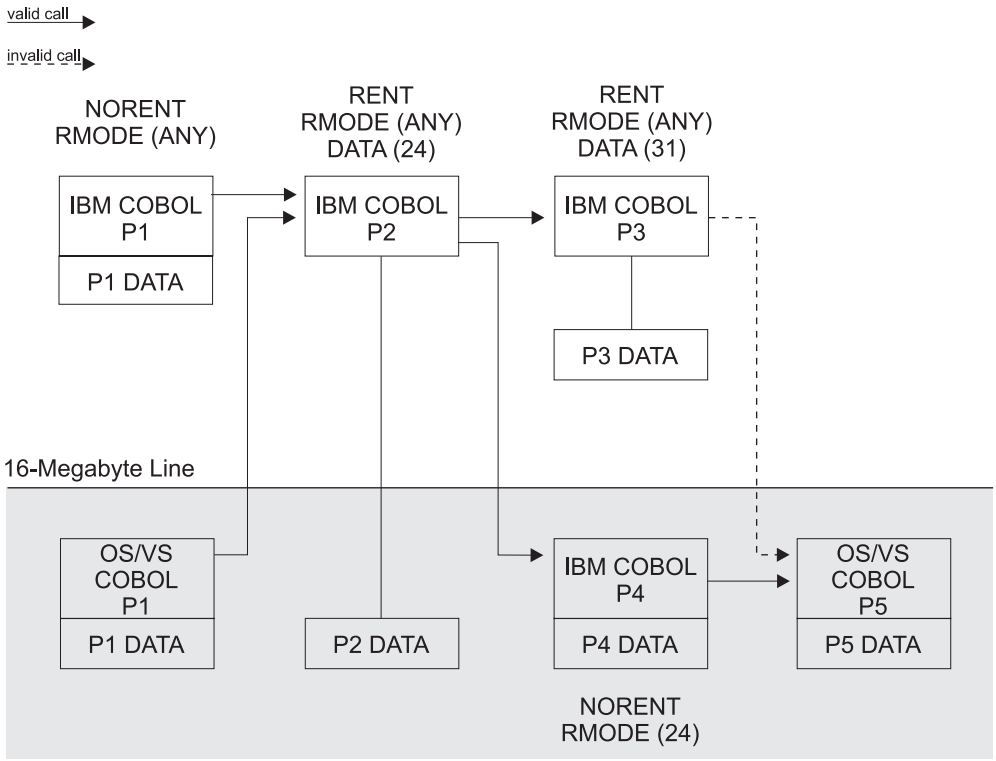


Figure 48. Valid and invalid CALLs between OS/VS COBOL and IBM COBOL programs

Run-time considerations

When adding COBOL for MVS & VM or COBOL for OS/390 & VM programs to existing applications (including applications consisting of COBOL/370 programs), you have additional considerations.

ILBOSRV: You must include the Language Environment Release 5 (or later) copy of ILBOSRV in the load module, in either of the following cases:

- If the existing load module contains and uses ILBOSTP0
- If the existing load module contains OS/VS COBOL NORES programs

TGT and RSA conventions: The TGT and RSA conventions in COBOL for OS/390 & VM are different than in COBOL for MVS & VM. If your existing IBM COBOL applications are coded to assume Register 9 or Register 13 values you must change those programs so they do not have built in assumptions about register values.

To find the TGT of a COBOL for MVS & VM or COBOL for OS/390 & VM program:

- When the COBOL for MVS & VM or COBOL for OS/390 & VM program is running, R13 has the address of a dynamic save area (DSA) for the COBOL program. The address of the COBOL program's TGT is located at x“5C” in the DSA.
- If R13 has the address of a save area associated with a called assembler program or COBOL run-time routine, use the save area back chain to find the DSA for the COBOL program. Then add x“5C” to find the address of the TGT.

  You can use the R15 slot in the previous save area to determine the entry point and signature information of the routine that owns the DSA.

Appendixes

Appendix A. Commonly asked questions and answers

This section provides answers to some of the most common questions about upgrading to IBM COBOL and Language Environment. The questions are grouped into the following categories:

- Prerequisites
- Compatibility
- Link-editing with Language Environment
- Compiling with IBM COBOL
- Language Environment services
- Language Environment run-time options
- Interlanguage communication
- Subsystems
- OS/390
- Performance
- Service

Prerequisites

► ***Can you use IBM COBOL with CICS Version 2?***

No. IBM COBOL compiled programs require Language Environment to run and Language Environment requires CICS Version 3 Release 3 or later.

► ***Can you use Language Environment with versions of CICS prior to Version 3 if the applications being run are VS COBOL II or OS/VS COBOL applications?***

No. Language Environment requires CICS Version 3 Release 3 even if the applications were created using previous versions of CICS and COBOL.

► ***Do you have to convert all macro-level COBOL programs to CICS Version 3 before they can be run with Language Environment?***

Yes. You can only run Language Environment in CICS Version 3, so any programs that you want to run under Language Environment must be upgraded to CICS Version 3.

► ***Is Language Environment required to run a program compiled with IBM COBOL?***

Yes, Language Environment contains the library routines required to run an IBM COBOL compiled program. COBOL/370, COBOL for MVS & VM, and COBOL for OS/390 & VM are compilers, and do not contain run-time library routines.

Compatibility

- ***I have a mix of COBOL and assembler programs. Do I need to change my assembler programs to be Language Environment enabled?***

  No, you do not need to change your assembler programs to be Language Environment enabled, however your assembler programs must follow S/390 save area conventions, namely:

  - The first halfword of the assembler save area must be hex zero
  - The back chain address must be set to the caller's save area
  - The back chain address must be a valid 31-bit address

- ***Can you run DOS/VS COBOL programs under Language Environment?***

  You can run programs compiled with DOS/VS COBOL under IBM Language Environment for VSE/ESA. For migration information, see the *IBM COBOL for VSE/ESA Migration Guide*, GC26-8070.

  Programs compiled with DOS/VS COBOL cannot run under Program Number 5688-198 (Language Environment).

- ***Does Language Environment support both OS/VS COBOL LANGLVL(1) and LANGLVL(2) compiled programs?***

  Yes.

- ***When running OS/VS COBOL in compatibility mode with VS COBOL II or Language Environment, 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 OS/VS COBOL.

  Note, that with IBM COBOL, assembler programs cannot get the address of the TGT from R13.

- ***With VS COBOL II, we have had errors where an output DD was misspelled and a temporary file was created. This causes problems when it occurs with a large file for a one time program run. Is this still a concern with IBM COBOL?***

  No, for QSAM you can turn off automatic file creation with the Language Environment CBLQDA(OFF) run-time option.

- ***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 OS/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 COBOLs.

- ***Can you place Language Environment in LNKLST/LPALST?***

  Before placing Language Environment in LNKLST/LPALST, test all applications that might access the Language Environment library routines from the LNKLST/LPALST. The move to Language Environment should be staged in a controlled manner.

  For more details, see “Decide how to phase Language Environment into production mode” on page 26.

- ***Can Language Environment coexist on a system with OS/VS COBOL and VS COBOL II?***

  Yes, but be aware of which library is needed and used for different applications. There are duplicate names between the different product libraries, so it is important to ensure the correct library is being accessed.

  For details, see “Invoke existing applications” on page 51.

- ***Is it easier to convert from OS/VS COBOL to IBM COBOL or from OS/VS COBOL to VS COBOL II?***

  The source conversion effort is exactly the same. Moving to the Language Environment run time is slightly more difficult if you have assembler programs that use SVC LINK or condition handling with OS/VS COBOL. Since the majority of time is spent testing, the two different conversion paths are approximately the same.

  For a summary of run-time considerations, see:

  - Figure 14 on page 23 for programs running under the OS/VS COBOL run time
  - Figure 15 on page 24 for programs running under the VS COBOL II run time

- ***Have there been any statements about when either support or marketing of VS COBOL II will be discontinued?***

  Service for VS COBOL II Release 3.0, 3.1, and 3.2 has been discontinued.

  Service for VS COBOL II Release 4, will be discontinued as follows:

  - For MVS and VM – March 31, 2001
  - For VSE – April 24, 1998

- ***Is the signature area of IBM COBOL programs the same as for OS/VS COBOL and VS COBOL II?***

  No, but a map of the signature area is in the *IBM COBOL Programming Guide* and can be used to find out what compiler options were used to compile the module, when it was compiled, release level, etc.

Link-editing with Language Environment

- ***When is it necessary to link-edit applications with Language Environment to run under Language Environment?***

  For exact link-edit requirements, see:

  – "Determining which programs require link-edit" on page 59 for programs running under the OS/VS COBOL run time

  – "Determining which programs require link-edit" on page 71 for programs running under the VS COBOL II run time

- ***Under OS/VS COBOL, aren't some library routines always invoked dynamically, even if the OS/VS COBOL program is compiled with NORES? Do I need to link-edit with Language Environment in order for these library routines to be supported when running under Language Environment?***

  Yes, under OS/VS COBOL ILBOD01, ILBODBE, ILBOPRM, ILBOSND, ILBOSTN, and ILBOTC2 are always invoked dynamically (unless explicitly INCLUDED by link-edit). Language Environment provides support for these library routines, regardless of whether or not the program is link-edited with Language Environment.

- ***Can OS/VS COBOL and VS COBOL II programs CALL IBM COBOL programs?***

  On non-CICS, any CALLs between OS/VS COBOL, VS COBOL II, and IBM COBOL are supported.

  On CICS, IBM COBOL programs cannot call or be called by OS/VS COBOL programs. EXEC CICS LINK must be used instead. CALLs to and from VS COBOL II programs and IBM COBOL programs are allowed. For additional details, see the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.

  For a complete list of calls between COBOL and assembler (including whether they are supported or not when running under Language Environment), see "Run-time support for assembler/COBOL calls on non-CICS" on page 211 and "Run-time support for assembler/COBOL calls on CICS" on page 212.

- ***Can OS/VS COBOL NORES load modules CALL and be CALLed by IBM COBOL programs?***

  IBM COBOL load modules can CALL OS/VS COBOL NORES load modules, if the NORES load module has been link-edited with Language Environment. The OS/VS COBOL NORES load module must return control to the IBM COBOL load module.

  It is possible to have a OS/VS COBOL NORES load module with "dynamic" calls (that is, using an assembler program that loads and branches) to an IBM COBOL program, which can then do a COBOL dynamic CALL to subsequent programs.

- ***Can you convert programs selectively to IBM COBOL?***

  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 OS/VS COBOL programs and IBM COBOL programs in the same run unit. When converting applications containing OS/VS COBOL programs that are run on CICS, you must convert all the OS/VS COBOL programs in the run unit to IBM COBOL.

Compiling with IBM COBOL

- ***Can you compile programs written for OS/VS COBOL with IBM COBOL using the CMPR2 option?***

  Some OS/VS COBOL source programs might compile cleanly under IBM COBOL with the CMPR2 option. The CMPR2 option supports COBOL 74 Standard as implemented by VS COBOL II Release 2. OS/VS COBOL LANGLVL(2) programs might compile if they do not use any unsupported language elements or invalid language that might have run in OS/VS COBOL.

- ***Can you compile programs written for VS COBOL II with IBM COBOL?***

  Yes. For additional details, see “Upgrade your source to IBM COBOL” on page 14.

- ***Do you have to compile OS/VS COBOL and VS COBOL II programs with IBM COBOL to run them under Language Environment?***

  No. Most OS/VS COBOL programs and VS COBOL II programs (and mixes of the two) will run under Language Environment without the need to upgrade to IBM COBOL.

  For exact details on which programs must be upgraded to IBM COBOL, see: see:

  - “Determining which programs require upgrade” on page 60 for programs running under the OS/VS COBOL run time
  - “Determining which programs require upgrade” on page 72 for programs running under the VS COBOL II run time

- ***What utilities or tools can assist in converting OS/VS COBOL or VS COBOL II source to IBM COBOL source?***

  The following conversion tools, which you can order through IBM, can assist in converting OS/VS COBOL and VS COBOL II source to IBM COBOL source.

  1. The COBOL conversion aid (CCCA) 5648-B05, assists in converting OS/VS COBOL and VS COBOL II source to IBM COBOL source.
  2. The COBOL Report Writer Precompiler 5798-DYR assists in converting OS/VS COBOL Report Writer code.
  3. The CICS application migration Aid 5695-061 assists in converting CICS macro level code in OS/VS COBOL and VS COBOL II to CICS command level code.
  4. The Edge Portfolio Analyzer 5633-009 assists in taking an inventory of your existing OS/VS COBOL and VS COBOL II load module libraries.

- ***Does IBM COBOL meet the COBOL 85 Standard?***

  Yes, IBM COBOL supports all required modules of the COBOL 85 Standard at the highest level defined by the Standard. In addition, some of the optional modules are supported, including the Intrinsic Functions Module.

Language Environment services

► ***Can VS COBOL II programs CALL Language Environment callable services?***

You can use dynamic CALLs from VS COBOL II programs to Language Environment date/time callable services only. You cannot use dynamic CALLs from VS COBOL II programs to other Language Environment callable services, nor can you use static CALLs to any Language Environment callable services from VS COBOL II programs.

You can dynamically CALL the following Language Environment services from VS COBOL II programs (any release):

CEECBLDY	CEEGMT	CEESCEN
CEEDATE	CEEGMTO	CEESECI
CEEDATM	CEEISEC	CEESECS
CEEDAYS	CEELOCT	CEE3CTY
CEEDYWK	CEEQCEN	

► ***Can users call Language Environment callable services from BAL (assembler) programs if the programs are Language Environment-conforming assembler programs?***

Yes. Any Language Environment-conforming BAL routines can use Language Environment callable services. Non-Language Environment-conforming BAL routines **cannot** use Language Environment callable services. The *Language Environment Programming Guide* describes how to make your existing BAL programs into Language Environment-conforming BAL programs, using macros supplied by IBM with the Language Environment product.

► ***Can OS/VS COBOL programs use Language Environment callable services?***

No. OS/VS COBOL programs cannot use the Language Environment callable services directly, but OS/VS COBOL programs can CALL a IBM COBOL program to place CALLs to the services.

► ***Can condition handling be added to a pure OS/VS COBOL application by converting the main routine to IBM COBOL?***

Yes, with restrictions on recovery. For details, see “Converting programs that use ESTAE/ESPIE for condition handling” on page 212.

► ***What is "COBOL multitasking" and how does it relate to PL/I Multitasking?***

"COBOL Multitasking" is the support of multiple applications running at the same time in the same address space under separate TCBs. It cannot be initiated with COBOL, but can be initiated via assembler doing ATTACHs or with ISPF split screens under Language Environment.

PL/I can initiate multitasking using native language and manage the interaction between the separate tasks.

Language Environment run-time options

- ***Will lower HEAP storage values for COBOL performance affect C/370 performance?***

  Yes. If the C programs use a lot of MALLOCs, then C performance will be worse with lower HEAP storage values.

- ***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, tune the HEAP values to improve performance. Also, if the application has many automatic variables, the STACK values should also be tuned to improve performance.

- ***Does COBOL use STACK storage?***

  IBM COBOL programs use stack storage, especially for LOCAL-STORAGE data items. Other COBOL programs do not use STACK storage.

  COBOL run-time routines do use STACK storage.

- ***Does OS/VS COBOL running on Language Environment use HEAP storage?***

  No, OS/VS COBOL WORKING-STORAGE does not use HEAP storage.

- ***What do HEAP(KEEP) or LIBSTACK(KEEP) do? Does the KEEP suboption keep all of the HEAP or LIBSTACK storage or just the increments of extra storage that were obtained?***

  The KEEP suboption causes Language Environment to keep **all** of the storage obtained, including the initial and incremental amounts.

- ***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 Language Environment abends with its own abend code. If an error is not HANDLED the application will terminate so ERRCOUNT will have no effect.

Interlanguage communication

- ***Why aren't assembler language programs discussed under interlanguage communication in Language Environment manuals or in IBM presentations?***

  Assembler language is not considered a separate language by Language Environment. 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 Language Environment.

- ***Can OS/VS COBOL or VS COBOL II programs CALL Language Environment-conforming assembler programs?***

  The following CALLs are supported:

  – On non-CICS

    - Dynamic CALLs from OS/VS COBOL RES programs or VS COBOL II RES programs to Language Environment-conforming assembler.
    - Static CALLs from OS/VS COBOL RES programs or VS COBOL II RES programs to Language Environment-conforming assembler. You must specify MAIN=NO and NAB=NO on the CEEENTRY macro.

  – On CICS

    - Dynamic CALLs from VS COBOL II to Language Environment-conforming assembler.

  For a complete list of calls between COBOL and assembler (including whether they are supported or not when running under Language Environment), see “Run-time support for assembler/COBOL calls on non-CICS” on page 211 and “Run-time support for assembler/COBOL calls on CICS” on page 212.

- ***Are there any problems with using TCP/IP and Language Environment together since TCP/IP requires the C/370 run time, and Language Environment includes the C/370 run time as well?***

  TCP/IP Version 3 Release 1 works with Language Environment Release 3 and later. Prior to these release levels, you could not use TCP/IP and Language Environment together.

Subsystems

- ***Does IBM COBOL have a method to return the century information from the system like CICS does? We need this for proper handling in DB2.***

  IBM COBOL and Language Environment each provide a method in which to handle the year 2000 issue. With IBM COBOL, you can use the CURRENT-DATE intrinsic function. With Language Environment, you can use the date and time callable services.

- ***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 CBLIDLI. 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.)

- ***Is CALL 'CEETDLI' supported in a CICS program? What about CALL 'CBLTDLI' in a CICS program running under Language Environment?***

  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 Language Environment.

- ***If you have a batch or IMS/DC application that has explicit calls to other Language Environment services, or user-coded Language Environment condition handlers, must all IMS interfaces use CEETDLI instead of CBLTDLI?***

  No, all calls within a program or run unit are not required to be CEETDLI. The exception is if you have any current application using the AIBTDLI interface. AIBTDLI should be changed to CEETDLI as it improves ESTAE processing and does not require a logic change, only a change to the call from AIBTDLI to CEETDLI.

- ***Will Language Environment (and its support of mixed COBOL and PL/I programs) still support applications with PL/I and VS COBOL II (or IBM COBOL) where the COBOL programs use CBLTDLI, or must such programs be converted to CEETDLI?***

  There is no problem with a mixed environment from an IMS standpoint and the programs do not need to be modified. Consider CBLTDLI and CEETDLI equivalent for conversion purposes.

  Under Language Environment, your COBOL programs can still use the CBLTDLI interface. Remember the programs must be VS COBOL II or IBM COBOL since mixed OS/VS COBOL and PL/I is not allowed under Language Environment. Either CBLTDLI or CEETDLI can be used, except that CEETDLI is not supported under a CICS environment.

- ***Do I need to specify the TRAP(OFF) run-time option when using the CBLTDLI interface under IMS?***

  No, TRAP(OFF) is never recommended for COBOL programs. There are some instances when you cannot use Language Environment condition handling when using CBLTDLI under IMS. However, if you specify ABTERMENC(ABEND), database rollback will be performed automatically for severe error conditions. For details, see the *Language Environment Programming Guide*.

- ***I am running both OS/VS COBOL and VS COBOL II programs on CICS. All of the VS COBOL II programs are AMODE31. Do I have to run with the Language Environment run-time option ALL31(OFF) because I have OS/VS COBOL programs (which are AMODE 24)?***

  You can run with ALL31(ON) if all of your VS COBOL II programs are AMODE 31. On CICS, OS/VS COBOL programs run in their own special compatibility environment and they are not affected by the Language Environment run-time options.

- ***Is IGZEDT4 provided in Language Environment?***

  Yes. However, for Language Environment Release 5, you need to apply apar PN84080.

OS/390

► ***Am I required to put Language Environment in the LNKLST with OS/390?***

No, but Language Environment must be installed in the same zone as OS/390. If you do not put Language Environment in the LNKLST, you must STEPLIB Language Environment in the individual OS/390 PROCs that require Language Environment.

For information on which elements require Language Environment, see:

– The OS/390 Program Directory for OS/390 Release 4 or later.

– Information APAR II10425 for OS/390 Release 1, 2, and 3.

► ***Can OS/VS COBOL programs run with the Language Environment element of OS/390?***

Yes. However, in some instances, link-editing with Language Environment is required. Other factors might apply as well. For details, see:

- Chapter 5, “Running existing applications under Language Environment” on page 46
- Chapter 6, “Moving from the OS/VS COBOL run time” on page 58

Performance

► ***Is it true there is a CPU savings when converting from OS/VS COBOL to IBM COBOL?***

IBM COBOL performance compared to OS/VS COBOL or VS COBOL II varies, depending on the characteristics of the applications. Information on COBOL for MVS & VM performance, is available on the web, in the Library Section, at:

`http://www.ibm.com/software/ad/cobol`

COBOL/370, COBOL for MVS & VM, and COBOL for OS/390 & VM performance are approximately the same. You might see improved performance in COBOL for MVS & VM and COBOL for OS/390 & VM for static and dynamic CALLs.

Service

► ***Do I need to recompile all of my programs to get IBM service support for my applications?***

As long as your programs are running with a supported run time you do not need to recompile your programs to continue to have IBM service support. For additional details, see “Service support for OS/VS COBOL and VS COBOL II programs” on page 3.

Appendix B. COBOL reserved word comparison

This appendix contains a table showing differences between OS/VS COBOL, VS COBOL II, and IBM COBOL reserved words. Information on source language comparison can be found in Chapter 10, “Modifying OS/VS COBOL source programs” on page 106.

This list identifies the reserved words in IBM COBOL, VS COBOL II, and OS/VS COBOL.

Note: Reserved words for COBOL for MVS & VM and COBOL for OS/390 & VM are included in the first column (marked 'IBM COBOL'). New reserved words, those which have been added since VS COBOL II, are highlighted in **boldface** type. The reserved word list is the same for COBOL for MVS & VM and COBOL for OS/390 & VM.

For COBOL/370 programs, only FUNCTION and PROCEDURE-POINTER are new reserved words (from the list of words in boldface type).

Key:

X	The word is reserved in the product.
X*	The word is newly reserved in COBOL for OS/390 & VM V2R2. (It is not reserved in V2R1 or earlier versions.)
-	The word is *not* reserved in the product. (This includes obsolete reserved words that are no longer flagged.)
CDW	The word is an IBM COBOL compiler directing statement. If used as a user-defined word, it is flagged with a severe message.
RFD	The word is reserved for future development. If used, it is 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 49 (Page 1 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
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	-

Figure 49 (Page 1 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
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

Figure 49 (Page 2 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
ARITHMETIC	RFD	RFD	-
ASCENDING	X	X	X
ASSIGN	X	X	X
AT	X	X	X
AUTHOR	X	X	X
B-AND	RFD	RFD	-
B-EXOR	RFD	RFD	-
B-LESS	RFD	RFD	-
B-NOT	RFD	RFD	-
B-OR	RFD	RFD	-
BASIS	CDW	CDW	X
BEFORE	X	X	X
BEGINNING	X	X	X
BINARY	X	X	-
BIT	RFD	RFD	-
BITS	RFD	RFD	-
BLANK	X	X	X
BLOCK	X	X	X
BOOLEAN	RFD	RFD	-
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	-
CLASS-ID	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

Figure 49 (Page 2 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
COLUMN	UNS	UNS	X
COM-REG	X	X	-
COMMA	X	X	X
COMMIT	RFD	RFD	-
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	X*	RFD	-
COMP-6	RFD	RFD	-
COMP-7	RFD	RFD	-
COMP-8	RFD	RFD	-
COMP-9	RFD	RFD	-
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	X*	RFD	-
COMPUTATIONAL-6	RFD	RFD	-
COMPUTATIONAL-7	RFD	RFD	-
COMPUTATIONAL-8	RFD	RFD	-
COMPUTATIONAL-9	RFD	RFD	-
COMPUTE	X	X	X
CONFIGURATION	X	X	X
CONNECT	RFD	RFD	-
CONSOLE	SYS	SYS	X
CONTAINED	RFD	RFD	-
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

Figure 49 (Page 3 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
CORR-INDEX	-	-	X
CORR	X	X	X
CORRESPONDING	X	X	X
COUNT	X	X	X
CSP	SYS	SYS	X
CURRENCY	X	X	X
CURRENT	RFD	RFD	-
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	RFD	RFD	-
DB-ACCESS-CONTROL-KEY	RFD	RFD	-
DB-DATA-NAME	RFD	RFD	-
DB-EXCEPTION	RFD	RFD	-
DB-RECORD-NAME	RFD	RFD	-
DB-SET-NAME	RFD	RFD	-
DB-STATUS	RFD	RFD	-
DBCS	X	X	-
DE	UNS	UNS	X
DEBUG	-	-	X
DEBUG-CONTENTS	X	X	X
DEBUG-ITEM	X	X	X

Figure 49 (Page 3 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
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	RFD	RFD	-
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	RFD	RFD	-
DISP	-	-	X
DISPLAY-ST	-	-	X
DISPLAY-1	X	X	-
DISPLAY-2	RFD	RFD	-
DISPLAY-3	RFD	RFD	-
DISPLAY-4	RFD	RFD	-
DISPLAY-5	RFD	RFD	-
DISPLAY-6	RFD	RFD	-
DISPLAY-7	RFD	RFD	-
DISPLAY-8	RFD	RFD	-
DISPLAY-9	RFD	RFD	-
DIVIDE	X	X	X
DIVISION	X	X	X
DOWN	X	X	X
DUPLICATE	RFD	RFD	-
DUPLICATES	X	X	X
DYNAMIC	X	X	X
EGCS	X	X	-
EGI	UNS	UNS	X
EJECT	CDW	CDW	X

Figure 49 (Page 4 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
ELSE	X	X	X
EMI	UNS	UNS	X
EMPTY	RFD	RFD	-
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	RFD	RFD	-
END-DIVIDE	X	X	-
END-ENABLE	RFD	RFD	-
END-EVALUATE	X	X	-
END-EXEC	X*	-	-
END-IF	X	X	-
END-INVOKE	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	RFD	RFD	-
END-START	X	X	-
END-STRING	X	X	-
END-SUBTRACT	X	X	-
END-TRANSCEIVE	RFD	RFD	-
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	RFD	RFD	-

Figure 49 (Page 4 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
ERASE	RFD	RFD	-
ERROR	X	X	X
ESI	UNS	UNS	X
EVALUATE	X	X	-
EVERY	X	X	X
EXACT	RFD	RFD	-
EXAMINE	-	-	X
EXCEEDS	RFD	RFD	-
EXCEPTION	X	X	X
EXCLUSIVE	RFD	RFD	-
EXEC	X*	-	-
EXHIBIT	-	-	X
EXIT	X	X	X
EXTEND	X	X	X
EXTERNAL	X	X	-
FACTORY	X*	-	-
FALSE	X	X	-
FD	X	X	X
FETCH	RFD	RFD	-
FILE	X	X	X
FILE-CONTROL	X	X	X
FILE-LIMIT	-	-	X
FILE-LIMITS	-	-	X
FILLER	X	X	X
FINAL	UNS	UNS	X
FIND	RFD	RFD	-
FINISH	RFD	RFD	-
FIRST	X	X	X
FOOTING	X	X	X
FOR	X	X	X
FORMAT	RFD	RFD	-
FREE	RFD	RFD	-
FROM	X	X	X
FUNCTION	X	RFD	-
GENERATE	UNS	UNS	X
GET	RFD	RFD	-
GIVING	X	X	X
GLOBAL	X	X	-

Figure 49 (Page 5 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
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	RFD	RFD	-
INDEX-2	RFD	RFD	-
INDEX-3	RFD	RFD	-
INDEX-4	RFD	RFD	-
INDEX-5	RFD	RFD	-
INDEX-6	RFD	RFD	-
INDEX-7	RFD	RFD	-
INDEX-8	RFD	RFD	-
INDEX-9	RFD	RFD	-
INDEXED	X	X	X
INDICATE	UNS	UNS	X
INHERITS	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
INVOKE	X	-	-
IS	X	X	X

Figure 49 (Page 5 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
JUST	X	X	X
JUSTIFIED	X	X	X
KANJI	X	X	-
KEEP	RFD	RFD	-
KEY	X	X	X
LABEL	X	X	X
LAST	UNS	UNS	X
LD	RFD	RFD	-
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
LOCAL-STORAGE	X	-	-
LOCALLY	RFD	RFD	-
LOCK	X	X	X
LOW-VALUE	X	X	X
LOW-VALUES	X	X	X
MEMBER	RFD	RFD	-
MEMORY	X	X	X
MERGE	X	X	X
MESSAGE	UNS	UNS	X
METACLASS	X	-	-
METHOD	X	-	-
METHOD-ID	X	-	-
MODE	X	X	X
MODIFY	RFD	RFD	-
MODULES	X	X	X
MORE-LABELS	X	X	X
MOVE	X	X	X

Figure 49 (Page 6 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
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	RFD	RFD	-
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	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	RFD	RFD	-
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
OVERRIDE	X	-	-
OWNER	RFD	RFD	-
PACKED-DECIMAL	X	X	-
PADDING	X	X	-
PAGE	X	X	X

Figure 49 (Page 6 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
PAGE-COUNTER	UNS	UNS	X
PARAGRAPH	RFD	RFD	-
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	RFD	RFD	-
PRINT-SWITCH	-	-	X
PRINTING	UNS	UNS	-
PRIOR	RFD	RFD	-
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	RFD	RFD	-
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	RFD	RFD	-
RECEIVE	UNS	UNS	X
RECONNECT	RFD	RFD	-
RECORD	X	X	X
RECORD-NAME	RFD	RFD	-

Figure 49 (Page 7 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
RECORD-OVERFLOW	-	-	X
RECORDING	X	X	X
RECORDS	X	X	X
RECURSIVE	X	-	-
REDEFINES	X	X	X
REEL	X	X	X
REFERENCE	X	X	-
REFERENCES	X	X	X
RELATION	RFD	RFD	-
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	RFD	RFD	-
REPLACE	X	X	-
REPLACING	X	X	X
REPORT	UNS	UNS	X
REPORTING	UNS	UNS	X
REPORTS	UNS	UNS	X
REPOSITORY	X	-	-
REREAD	-	-	X
RERUN	X	X	X
RESERVE	X	X	X
RESET	-	-	X
RETAINING	RFD	RFD	-
RETRIEVAL	RFD	RFD	-
RETURN	X	X	X
RETURNING	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

Figure 49 (Page 7 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
RIGHT	X	X	X
ROLLBACK	RFD	RFD	-
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
SELF	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	RFD	RFD	-
SET	X	X	X
SHARED	RFD	RFD	-
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

Figure 49 (Page 8 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
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
SQL	X*	-	-
STANDARD	X	X	X
STANDARD-1	X	X	X
STANDARD-2	X	X	-
STANDARD-3	RFD	RFD	-
STANDARD-4	RFD	RFD	-
START	X	X	X
STATUS	X	X	X
STOP	X	X	X
STORE	RFD	RFD	-
STRING	X	X	X
SUB-QUEUE-1	UNS	UNS	X
SUB-QUEUE-2	UNS	UNS	X
SUB-QUEUE-3	UNS	UNS	X
SUB-SCHEMA	RFD	RFD	-
SUBTRACT	X	X	X
SUM	UNS	UNS	X
SUPER	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

Figure 49 (Page 8 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
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	RFD	RFD	-
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	RFD	RFD	-
TRANSFORM	-	-	X
TRUE	X	X	-
TYPE	X	-	-
UNEQUAL	RFD	RFD	-
UNIT	X	X	X
UNSTRING	X	X	X

Figure 49 (Page 9 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
UNTIL	X	X	X
UP	X	X	X
UPDATE	RFD	RFD	-
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	RFD	RFD	-
USE	X	X	X
USING	X	X	X
VALID	RFD	RFD	-
VALIDATE	RFD	RFD	-
VALUE	X	X	X
VALUES	X	X	X
VARYING	X	X	X
WAIT	RFD	RFD	-
WHEN	X	X	X
WHEN-COMPILED	X	X	X
WITH	X	X	X
WITHIN	RFD	RFD	-
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

Figure 49 (Page 9 of 9). Reserved words

Reserved word	IBM COBOL	VS II	OS/VS
-	X	X	X
/	X	X	X
>	X	X	X
>=	X	X	-
=	X	X	X

Appendix C. Conversion tools for source programs

This appendix describes the conversion tools available for your assistance during the actual conversion activity. These tools are:

- CMPR2, FLAGMIG, and NOCOMPILE compiler options
- MIGR compiler option (OS/VS COBOL)
- Other programs that aid conversion

This appendix helps you to determine which, if any, of the conversion tools to use, understand how to use them, and how to analyze the conversion tool output to assess the extent of the remaining conversion effort.

Note: All of these conversion tools assume that the program you are converting is a valid OS/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 OS/VS* and *VS COBOL II Application Programming Language Reference*, with no undocumented extensions.

CMPR2, FLAGMIG, and NOCOMPILE compiler options

By compiling existing OS/VS COBOL application programs with the IBM COBOL compiler, you can identify some of the OS/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, as well as nonstandard Release 2 extensions now implemented in the COBOL 85 Standard.

FLAGMIG Identifies OS/VS COBOL and VS COBOL II language that is incompatible with IBM COBOL 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 IBM COBOL NOCMPR2.

NOCOMPILE Identifies any language not accepted by the IBM COBOL compiler without producing an object deck, thus saving system resources over a full compile with the COMPILE compiler option.

MIGR compiler option

You can use the OS/VS COBOL MIGR compiler option whenever you are planning to convert an OS/VS COBOL program to IBM COBOL 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 OS/VS COBOL source language not supported by IBM COBOL 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 because of added COBOL functions (previously valid user words might now be illegal)

- 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 an OS/VS COBOL program. When you set MIGR on, the compiler flags most statements that are changed in or not supported by IBM COBOL with NOCMPR2.

Language differences

The following language differences exist between IBM COBOL and OS/VS COBOL.

- ALPHABETIC class changes
- B symbol in PICTURE clause
- CALL statement changes
- CBL compiler directing statement changes
- Combined abbreviated relation condition changes
- DIVIDE ID1 BY ID2 [GIVING ID3] ON SIZE ERROR ...
- DIVIDE ID1 INTO ID2 [GIVING ID3] ON SIZE ERROR ...
- 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 ID1 BY ID2 [GIVING ID3] ON SIZE ERROR ...
- 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
- SPECIAL-NAMES: alphabet-name IS xxxxx
- Subscripts out of range—changes in evaluation
- 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

Statements supported with enhanced accuracy

Following are OS/VS COBOL statements supported with enhanced accuracy in IBM COBOL and flagged by a message indicating that more accurate results might be provided in IBM COBOL.

Arithmetic statements

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

LANGLVL(1) statements not supported

The following OS/VS COBOL statements, applicable only to the LANGLVL(1) compiler option, are not supported in IBM COBOL and are flagged when the MIGR compiler option is specified.

- COPY language—1968
- JUSTIFIED|JUST clause with VALUE
- MOVE statement and comparison—scaling changes
- NOT in an abbreviated combined relation condition
- PERFORM statement in independent segments
- RESERVE integer AREAS
- SELECT OPTIONAL clause—1968 standard interpretation
- SPECIAL-NAMES paragraph: use of L, /, and =
- UNSTRING with DELIMITED BY ALL

LANGLVL(1) and LANGLVL(2) statements not supported

The following OS/VS COBOL statements, applicable to both the LANGLVL(1) and LANGLVL(2) compiler options, are not supported in IBM COBOL and are flagged when the MIGR compiler option is specified.

Communications

- COMMUNICATION SECTION
- ACCEPT MESSAGE
- SEND, RECEIVE, ENABLE, and DISABLE verbs. (Note that RECEIVE ...MESSAGE is LANGLVL sensitive, but is flagged only under Communications.)

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 can convert these statements for you. See “COBOL Report Writer Precompiler” on page 207.

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

BDAM

- ACTUAL KEY clause
- APPLY RECORD-OVERFLOW (BDAM)
- BDAM file declarations
- I/O verbs—all that reference BDAM files
- Organization parameters “D”, “R”, and “W”
- SEEK statement
- TRACK-LIMIT clause

Use for debugging

- USE FOR DEBUGGING ON [ALL REFERENCES OF] identifiers, file-names, cd-names

Other statements

- APPLY RECORD-OVERFLOW
- Assignment-name organization parameter “C” indicating ASCII
- ASSIGN ... OR
- ASSIGN TO integer system-name
- ASSIGN ... FOR MULTIPLE REEL/UNIT
- CLOSE ... WITH POSITIONING/DISP
- CURRENT-DATE and TIME-OF-DAY special registers
- Debug packets
- EXAMINE statement
- EXHIBIT statement
- FILE-LIMITS
- LABEL RECORDS Clause with TOTALING/TOTALED AREA options
- NOTE statement
- ON statement
- OPEN ... LEAVE/REREAD/DISP
- Qualified index-names

  (Using this unsupported format will result in a severe (RC = 12) level message.)
- 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

Other programs that aid conversion

The following sections describe several conversion tools that offer you help in your conversion tasks. These programs are:

- On the workstation
  - Report Writer
- On the host:

- COBOL and CICS/VS Command Level Conversion Aid (CCCA)
- CICS application migration aid
- COBOL Report Writer Precompiler
- COBOL Structuring Facility (COBOL/SF)
- Vendor products

Report Writer for OS/2 and for Windows

As an optional, separately orderable feature of VisualAge for COBOL, Report Writer for OS/2 and for Windows delivers a general-purpose COBOL printing facility and provides the familiar host Report Writer function on the workstation.

For additional information, see "COBOL Report Writer Precompiler" on page 207.

COBOL and CICS/VS Command Level Conversion Aid (CCCA)

The COBOL and CICS/VS Command Level Conversion Aid (CCCA), Product Number 5648-B05, converts CICS and non-CICS source code into source code compilable by IBM COBOL.

The CCCA is designed to automate identifying incompatible source code and converting it to IBM COBOL source. Using CCCA should significantly reduce your conversion effort.

The CCCA requires that you have an IBM COBOL, VS COBOL II, or OS/VS COBOL compiles available when converting CICS programs.

The following are the key CCCA facilities:

- Conversion of most syntax differences between OS/VS COBOL or VS COBOL II programs and IBM COBOL programs
- Elimination of conflicts between OS/VS COBOL and VS COBOL II user-defined names and IBM COBOL 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

The CCCA is designed so that you can tailor it to fit the needs of your DP shop. 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/VS Command Level Conversion Aid* 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.

When to use CCCA

If you plan to convert your applications from OS/VS COBOL or VS COBOL II to IBM COBOL, evaluate the usefulness of the CCCA to your conversion project. While the number of changes required to any individual program might be small, the CCCA will identify those changes, and in the majority of cases convert them automatically in a standard fashion. The CCCA converts both CICS and non-CICS programs. The CCCA converts SERVICE RELOAD statements and the complicated logic of BLL cell addressing to statements valid for IBM COBOL.

CCCA also handles non-CICS syntax.

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 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 IBM COBOL 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.

Statements dealing with the primary BLLs

The statements dealing with the primary BLLs are shown in Figure 50.

Statements dealing with the secondary BLLs are replaced by CONTINUE.

Figure 50. 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 ...

CICS Application Migration Aid

The CICS Application Migration Aid, Product Number 5695-061, simplifies the conversion of COBOL and assembler language application programs from macro-level application programming interface (API) to the command-level API. Simpler macros are converted automatically, providing new source code with the equivalent command-level functions. More complex macros are partially converted, and guidance information is provided to aid you in completing the conversion. The original source code is still available in all cases.

Command-level is a requirement for applications that are to run on CICS/ESA Version 3 or later.

COBOL Report Writer Precompiler

The Report Writer Precompiler, product number 5798-DYR, has two functions. It can be used to precompile applications containing Report Writer statements so the code will be acceptable to the IBM COBOL compilers, or it can permanently convert Report Writer statements to valid IBM COBOL 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 IBM COBOL 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 IBM COBOL compiler

For more detail, see *COBOL Report Writer Precompiler Programmer's Manual* and *COBOL Report Writer Precompiler Installation and Operation*.

COBOL Structuring Facility (COBOL/SF)

The COBOL Structuring Facility (COBOL/SF), Product Number 5696-737, 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.

The input to COBOL/SF should be valid OS/VS COBOL, VS COBOL II, or IBM COBOL 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 reengineer programs repeatedly, to ensure that their structured code does not deteriorate during maintenance operations, but 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, DLI, or CICS commands. (There are restrictions when using COBOL/SF with certain CICS commands. For information on these restrictions, see the *COBOL Structuring Facility Reference*.

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.

The Edge Portfolio Analyzer

The Edge Portfolio Analyzer, Product Number 5620-AZM, helps you to take an inventory of your existing OS/VS COBOL and VS COBOL II load modules. The Edge Portfolio Analyzer can:

- Determine which version and release of the OS/VS COBOL compiler or the VS COBOL II compiler created the load module
- Determine which compiler options were specified when the load module was compiled
- Determine which load modules call for the current system date
- Determine which CSECTs need to be replaced, such as ILBOSRV.

Vendor products

A number of non-IBM conversion tools are available to help you upgrade your source programs to IBM COBOL programs and move your run time to Language Environment. IBM has compiled a list of vendor products enabled to work with Language Environment and IBM COBOL in the *Language Environment Enabled Vendor Tools and Application Packages* document. You can get this information: on the Web at `http://www.ibm.com/s390/le` then go to the Library link.

Appendix D. Applications with COBOL and assembler

This chapter contains information for applications that contain mixed COBOL programs and assembler programs. It includes information on:

- Determining requirements for calling and called assembler programs
- Determining which assembler/COBOL calls are supported on non-CICS
- Determining which assembler/COBOL calls are supported on CICS
- Converting programs that use ESTAE/ESPIE for condition handling
- Converting programs that change the program mask
- Upgrading applications that use an assembler driver
- Invoking a COBOL program with an MVS ATTACH
- Assembler programs that load and call COBOL programs
- Freeing storage in subpools
- Invoking programs - AMODE requirements

Some information on applications with both assembler programs and COBOL programs is included in other chapters of this book. Figure 51 list the information and page reference of where additional information on assembler programs is located.

Figure 51. Additional information about assembler programs and COBOL programs

Program attribute	Reference
Assembler routine using a LINK SVC or CMSCALL	Page 60 or 73
Assembler routine using a LINK SVC while in a reusable environment	Page 80
Assembler programs that pass procedure names	Page 132
Assembler programs that do not close files under MVS	Page 78
AMODE behavior when COBOL programs return to assembler programs	Page 102

Determining requirements for 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 2 bytes of the save area must be zero. The save area back chain must be set with a valid 31-bit address. (That is, the high-order byte must be cleared when running AMODE 24.)

The recommended way to establish a register save area address is:

```
          LA   13, SAVEAREA
          .
          .
          .
SAVEAREA  DS   18F
```

as this will ensure that the high-order byte is cleared for AMODE 24 programs.

If you must use a branch and link type of instruction to jump over the save area and set its address, the preferred method is:

```
          BAS  13,SKIP
SAVEAREA  DS   18F
SKIP      DS   0H
```

If you use a BAL instruction, you must clear the high-order byte yourself, as follows:

```
          BAL  13,SKIP
SAVEAREA  DS   18F
SKIP      LA   13,0(,13)
```

The BAL instruction puts the instruction length code, the condition code (CC), and the program mask in the high-order byte of the register. We recommend using the BAS instruction instead of the BAL instruction, since the BAS instruction places zeros in the high-order byte, thus preventing the 24-bit addressing problem.

If the program passes parameters, 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 pass a parameter list, it must be a group of one or more contiguous fullwords, each of which contain 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 COBOL 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

SVC LINK and COBOL run unit boundary

If the target of SVC LINK is a non-Language Environment-conforming assembler program, and the assembler program later calls a COBOL program, the Language Environment enclave and COBOL run unit boundary will be at the COBOL program, not at the assembler program. The main program of the enclave (and run unit) is the COBOL program.

If the target of SVC LINK is a Language Environment-conforming assembler program, the Language Environment enclave boundary will be at the assembler program. The assembler program is the main program of the enclave (provided MAIN=YES is specified in the CEEENTRY macro). If the assembler program calls a COBOL program at a later time, the COBOL program is a subprogram.

Run-time support for assembler/COBOL calls on non-CICS

Figure 52 lists the possible combinations of calls involving COBOL programs and assembler programs and indicates whether the calls are supported or not supported when running under Language Environment on non-CICS. For the calls that are **not** supported, this table also lists the symptom (message or abend code) that is returned in most cases. In some cases, depending on the application environment, the symptom might not occur. You could receive a different failure, or the application might appear to run successfully.

Note: In this table, **IBM COBOL** refers to COBOL/370, COBOL for MVS & VM and COBOL for OS/390 & VM.

Figure 52. Language Environment's support for calls between COBOL programs and assembler programs on non-CICS

Calls from		Issued to					
Program issuing	**Call type**	**IBM COBOL**	**VS II COBOL**	**OS/VS COBOL**	**LanEnv[1] assembler main**	**LanEnv[1] assembler subroutine**	**Non-LanEnv assembler**
IBM COBOL	Static	Yes	Yes	Yes	No[5]	Yes	Yes
VS COBOL II		Yes	Yes	Yes	No[5]	Yes[6]	Yes
OS/VS COBOL		Yes	Yes	Yes	No[5]	Yes[6]	Yes
IBM COBOL	Dynamic	Yes	Yes	Yes	No[5]	Yes	Yes
VS COBOL II		Yes	Yes	Yes	No[5]	Yes	Yes
OS/VS COBOL		Yes	Yes	Yes	No[5]	Yes	Yes
Assembler (LanEnv)	VCON	Yes	Yes	Yes	No[5]	Yes	Yes
Assembler (non-LanEnv)		Yes	Yes	Yes	Yes[4]	No[2]	Yes
Assembler (LanEnv)	LOAD	Yes	Yes	Yes	No[5]	Yes	Yes
Assembler (non-LanEnv)	BALR	Yes	Yes	Yes	Yes[4]	No[2]	Yes
Assembler (LanEnv)	LINK	Yes	Yes	Yes[3]	Yes	No[2]	Yes
Assembler (non-LanEnv)		Yes	Yes	Yes[3]	Yes	No[2]	Yes

Note: The failure symptoms described in these notes are as they would appear when the Language Environment TRAP(ON) and ABTERMENC(ABEND) run-time options are in effect.

1 CEEENTRY macro with MAIN=YES creates a Language Environment assembler main. If you specify MAIN=NO on the CEEENTRY macro, a Language Environment assembler subroutine is created. The default is MAIN=YES.

2 Failure symptom of: 0C1, 0C4, or 0C5 abend.

3 Except when OS/VS COBOL programs exist in another established Language Environment enclave. For detail, see “OS/VS COBOL programs in more than one enclave” on page 60 or “OS/VS COBOL programs in more than one enclave” on page 73. Failure symptom of: message IGZ0005S.

4 If the non-Language Environment assembler caller is running within an established Language Environment enclave, see note 5.

5 Invoking a Language Environment assembler main program from an established Language Environment enclave is not recommended (unless through the use of SVC LINK). For this reason, the table entries associated with this footnote are marked 'No'. A nested enclave is not created and, therefore, the program runs as a subprogram in the invoking enclave. If you follow this recommendation, you might avoid the need for reprogramming in the future.

6 You must specify NAB=NO and MAIN=NO on the CEEENTRY macro. Otherwise, you will receive failure symptom: 0C1, 0C4, or 0C5 abend.

Run-time support for assembler/COBOL calls on CICS

Figure 53 lists the possible combinations of calls involving COBOL programs and assembler programs and indicates whether the calls are supported or not when running under Language Environment on CICS. For the calls that are **not** supported, this table also lists the symptom (message or abend code) that will be returned in most cases. In some cases, depending on the application environment, the symptom might not occur; you could receive a different failure, or the application might appear to run successfully.

Note: In this table, **IBM COBOL** refers to COBOL/370, COBOL for MVS & VM, and COBOL for OS/390 & VM.

Figure 53. Language Environment's support for calls between COBOL programs and assembler programs that run on CICS

Calls from		Issued to					
Program issuing	**Call type**	**IBM COBOL**	**VS II COBOL**	**OS/VS COBOL**	**LanEnv[1] assembler main**	**LanEnv[1] assembler subroutine**	**Non-LanEnv assembler**
IBM COBOL	Static	Yes	Yes	No[3]	No[5]	Yes	Yes
VS COBOL II		Yes	Yes	No[3]	No[5]	No[2]	Yes
OS/VS COBOL		No[2]	No[2]	Yes	No[2]	No[2]	Yes
IBM COBOL	Dynamic	Yes	Yes	No[3]	No[5]	Yes	Yes
VS COBOL II		Yes	Yes	No[3]	No[5]	Yes	Yes
OS/VS COBOL		No[4]	No[4]	No[4]	No[4]	No[4]	No[4]
IBM COBOL	EXEC	Yes	Yes	Yes	No[5]	No[2]	Yes
VS COBOL II	CICS	Yes	Yes	Yes	No[5]	No[2]	Yes
OS/VS COBOL	LINK	Yes	Yes	Yes	No[5]	No[2]	Yes
Assembler (LanEnv)	VCON	Yes	No[2]	No[2]	No[5]	Yes	Yes
Assembler (non-LanEnv)		No[2]	No[2]	No[2]	No[5]	No[2]	Yes
Assembler (LanEnv)	EXEC CICS	Yes	Yes	Yes	No[5]	No[2]	Yes
Assembler (non-LanEnv)	LINK	Yes	Yes	Yes	No[5]	No[2]	Yes

Note: The failure symptoms described in these notes are as they would appear when the Language Environment TRAP(ON) and ABTERMENC(ABEND) run-time options are in effect.

1 CEEENTRY macro with MAIN=YES creates a Language Environment assembler main. If you specify MAIN=NO on the CEEENTRY macro, a Language Environment assembler subroutine is created. The default is MAIN=YES.

2 Failure symptom of: ASRA abend (caused by type 1 or 4 program check).

3 Failure symptom of: message IGZ0079S.

4 Failure symptom of: U3504 abend.

5 There is no support for Language Environment-conforming assembler main programs under CICS. Failure symptom: Unpredictable. The applications might appear to run successfully.

Converting programs that use ESTAE/ESPIE for condition handling

VS COBOL II and OS/VS COBOL applications can contain user-written condition handling routines written in assembler. Normally, you would code an assembler routine (this is your ESTAE routine) to set an ESTAE and register an ESTAE exit. When an abend occurs, the ESTAE exit receives control. (Although this section references ESTAE, this information also applies to assembler routines that set ESPIEs and register ESPIE exits.)

When running programs under Language Environment, the Language Environment condition manager receives control for errors, program interrupts, and abends.

Existing user-written condition handling routines do not work under Language Environment.

Error handling routines in existing programs: For VS COBOL II applications and OS/VS COBOL applications, total source conversion is not necessary. You only need to convert programs that either establish or perform condition handling.

However, if the application code is maintained in OS/VS COBOL or VS COBOL II, automatic stack frame collapse will not take place. If the application resumes after the condition handler gets control, you must explicitly CANCEL any programs that will be reentered before they are reentered.

To convert programs that establish or perform condition handling:

1. Replace the CALL to the ESTAE routine with a CALL to CEEHDLR (a Language Environment callable service). Only IBM COBOL programs can use CEEHDLR.

   If you want to resume and have more control of where to resume, use the SET RESUME POINT service (CEESRP). For details, see the *Language Environment Programming Reference*.

2. Make your ESTAE exit into a separate Language Environment-conforming routine (IBM COBOL, Language Environment-conforming assembler, or Language Environment-conforming C).

3. Change the logic of your condition handler. You must use Language Environment 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 Language Environment services to find the name of the program that incurred the condition, or retrieve the error message associated with the condition.

Establishing user-written condition handling routines: Using IBM COBOL's PROCEDURE-POINTER data item (an IBM extension to the COBOL 85 Standard) in conjunction with the SET statement, you can establish your own condition handling routine using Language Environment-provided callable services. The user-written condition handling routines receive control before the Language Environment default condition handling. You can write user-written condition handling routines in either IBM COBOL, Language Environment-conforming C, or Language Environment-conforming assembler.

Advantages user-written condition handling routines: Managing the point where you resume execution after handling an error is much simpler with Language Environment, than with OS/VS COBOL or VS COBOL II. Under OS/VS COBOL, VS COBOL II, and Language Environment, recursive CALLs are not allowed. For example, under OS/VS COBOL and VS COBOL II, if Program A CALLs Program B and an error occurs in Program B thus passing control back to Program A, Program A then cannot CALL Program B since this would cause a recursive error. Thus, after intercepting an error, the condition handler must resume at the next sequential instruction (NSI) following the instruction that incurred the error.

With OS/VS COBOL and VS COBOL II, to resume in a program other than the one that incurred the error, you are required to CANCEL the programs that are active at the time of the error and that might be reentered after the ESTAE exit received control.

When running IBM COBOL or Language Environment-conforming assembler programs under Language Environment, Language Environment automatically deactivates your programs when you change the resume point. Applying the above example to programs run under Language Environment, after resuming in Program A, you **can** call Program B. Language Environment has deactivated Program B, so no recursion occurs.

For VS COBOL II programs, the move resume cursor function (Language Environment callable service CEEMRCR) will cause stack frame collapse, where programs that were active are rendered inactive, and can then be reentered (except for VS COBOL II programs compiled with NOCMPR2 that use nested programs).

For VS COBOL II programs that can be reentered, you can:

- Resume in the 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.

Converting programs that change the program mask

When a VS COBOL II program calls an assembler program that changes the program mask (for example, uses an SPM instruction), the program mask is restored after the call to the assembler program.

With COBOL/370 and IBM COBOL, the program mask is not restored. Thus, if you change the program mask in your assembler program, you must restore it before returning to the COBOL program. Failure to restore the program mask could result in undetected data errors, such as fixed-point overflow, decimal overflow, exponent underflow, and significance exceptions.

Calling assembler programs that expect a certain program mask

With VS COBOL II, the program mask always had the same setting when an assembler program was called. Now with Language Environment, the program mask will be unpredictable when called from COBOL. If the assembler program has certain program mask requirements, it needs to set the program mask to what it needs, then restore the program mask before returning.

Upgrading applications that use an assembler driver

There are three methods you can use to upgrade applications that use an assembler driver to call COBOL subroutines:

- Convert assembler driver to a Language Environment-conforming assembler driver
- Modify the driver to set up the Language Environment environment
- Use the RTEREUS run-time option if the driver cannot be modified

These methods are described in the sections below. (In all cases, you upgrade the COBOL subroutines in the same way as described in the other COBOL conversion scenarios.)

Convert assembler driver: To upgrade an application that has an assembler driver, you can change the assembler driver to be a Language Environment-conforming assembler main program. For details on how to make your existing assembler programs Language Environment-conforming, see the *Language Environment Programming Guide*.

Modify assembler driver: If you want to modify the assembler driver routine, you can replace the OS/VS COBOL ILBOSTP0 routine with the Language Environment IGZERRE INIT and IGZERRE TERM functions. These Language Environment routines have a convenient complementary termination function not available with OS/VS COBOL.

If either IGZERRE or ILBOSTP0 is statically linked with the assembler driver, the assembler driver must be link-edited with Language Environment. If ILBOSTP0 is used, you must specify Language Environment options: ALL31(OFF) and STACK(,,BELOW).

For alternative methods to use preinitialization, see “Initializing the run-time environment” on page 92.

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 Language Environment RTEREUS run-time option. (RTEREUS initializes the run-time environment for reusability when the first COBOL program is invoked.)

Important: RTEREUS is not recommended for all applications; in some instances, it exhibits undesirable behavior. Before using RTEREUS, thoroughly explore the possible side-effects and understand the impact on your application. For more information, see “Recommended run-time options for non-CICS applications” on page 46.

Invoking a COBOL program with an MVS ATTACH

When you invoke a COBOL main program using an MVS ATTACH, Language Environment processes parameter lists differently than VS COBOL II.

Under Language Environment, when a COBOL program is invoked directly using ATTACH SVC (including the invocation of a batch program by the operating system and invocation from TSO CALL/ATTACH), the parameter list is always processed as a "PARM=" style.

Under VS COBOL II, when a COBOL program is invoked directly using ATTACH SVC (including the invocation of a batch program by the operating system and invocation from TSO CALL/ATTACH), the parameter list is processed as a "PARM=" style only when:

- Register 1 is nonzero.
- The word addressed by register 1 (the first parameter pointer word) has the End of LIST (EOL) bit on.
- The parameter addressed by the EOL bit is aligned on a halfword or greater boundary.

Otherwise, register 1 and the parameter list are passed without change.

There are two ways to get compatible behavior:

1. Change the main program to Language Environment-conforming assembler, and use PLIST=OS keyword in the CEEENTRY macro. Then have the assembler program call the COBOL program. See the sample code below on how to do this.

```
ASMLE3   CEEENTRY PPA=MAINPPA,AUTO=WORKSIZE,MAIN=YES,PLIST=OS
         USING WORKAREA,13
         L     15,A1C401P         Get the addr of the COBOL pgm
         BALR  14,15              Call it with parm list unchanged
*=================================================================
*   Terminate Language Environment.
*=================================================================
         CEETERM  RC=0
MAINPPA  CEEPPA                   Constants describing the code block
*=================================================================
*        The Workarea and DSA
*=================================================================
A1C401P  DC    V(A1C401P)         VCON FOR COBOL pgm
WORKAREA DSECT
         ORG    *+CEEDSASZ        Leave space for the DSA fixed part
         DS     0D
WORKSIZE EQU    *-WORKAREA
         CEEDSA                   Mapping of the Dynamic Save Area
         CEECAA                   Mapping of the Common Anchor Area
         CEEEDB                   Mapping of the Enclave Data Block
         END   ASMLE3
```

2. You can modify Language Environment to ensure that the parameter list processing for a COBOL main program invoked using an ATTACH has the same behavior as when the COBOL program runs with the VS COBOL II run time. To modify the parameter list processing under Language Environment, run the sample customization job IGZWAPSX with a modified copy of IGZEPSX (the COBOL parameter list exit routine).

   For instructions on how to modify IGZEPSX see:

   - For OS/390: *Language Environment for OS/390 Customization*
   - For MVS (Language Environment Release 5): APAR PN78697
   - For VM: *Program Directory*

Assembler loading and calling COBOL programs

COBOL programs that are compiled RENT, called from assembler, and then dynamically CALLed from COBOL behave differently under Language Environment than under the VS COBOL II run time.

For VS COBOL II and IBM COBOL programs that are:

1. Compiled with the RENT option
2. Dynamically CALLed from an OS/VS COBOL, VS COBOL II, COBOL/370 or IBM COBOL program
3. Loaded and called by assembler
4. Run under Language Environment Release 5 or later

the same copy of WORKING-STORAGE is used for each call. And, the program is entered in its last-used state, unless there is an intervening CANCEL.

Under VS COBOL II, different copies of WORKING-STORAGE is used for each call.

Assembler programs that load and delete COBOL programs

With Language Environment, assembler programs can SVC load and SVC delete load modules that contain any of the following:

- OS/VS COBOL programs
- VS COBOL II programs compiled with the NORENT option
- COBOL/370 programs compiled with the NORENT option
- IBM COBOL programs compiled with the NORENT option

Note: The Debug Tool does not support COBOL programs that are in load modules that are deleted by assembler using SVC delete.

With Language Environment, assembler programs can SVC load but **cannot** SVC delete load modules that contain any of the following:

- VS COBOL II programs compiled with the RENT option
- COBOL/370 programs compiled with the RENT option
- IBM COBOL programs compiled with the RENT option

If assembler programs SVC delete load modules that contain these kinds of programs, unpredictable results can occur.

For assembler programs that need to load and delete load modules that contain a COBOL RENT program, do one of the following:

- Have the assembler program statically call a COBOL program that performs the dynamic CALL and performs the CANCEL.
- Under Language Environment Release 7 (OS/390 Release 3) or later, use the CEEFETCH and the CEERELES macros.

Note: CEEFETCH can be used to load Language Environment-conforming programs only. CEEFETCH can be used to load COBOL/370, COBOL for MVS & VM, and COBOL for OS/390 & VM programs. CEEFETCH cannot be used to load VS COBOL II or OS/VS COBOL programs.

Freeing storage in subpools (OS/390 and MVS only)

On OS/390 and MVS, Language Environment allocates storage out of subpool 1 and subpool 2. VS COBOL II only used subpool 0. If you free storage in subpool 1 or subpool 2, you can lose data placed in those subpools by Language Environment.

Invoking programs - AMODE requirements

With VS COBOL II, assembler programs could invoke COBOL programs regardless of the AMODE specification. For example:

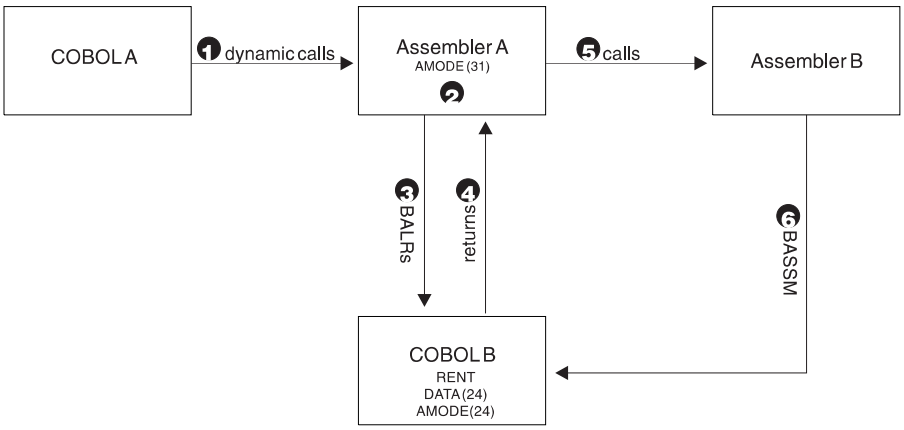


Figure 54. Effect of AMODE for invoking programs

In the figure above, the following occurs:

1. Program `COBOLA` dynamically CALLs `AssemblerA` (Assembler is AMODE 31.)
2. `AssemblerA` loads `COBOLB` COBOLB is RENT, DATA(24), and AMODE(24).
3. `AssemblerA` BALRs to `COBOLB`. (Now, COBOLB is entered in AMODE 31.)
4. `COBOLB` returns to `AssemblerA`.
5. `AssemblerA` calls `AssemblerB`.
6. `AssemblerB` does a BASSM to `COBOLB`. (COBOLB is entered in AMODE 24.) COBOLB abends because it expects to be entered in AMODE 31.

With Language Environment, COBOL programs compiled with VS COBOL II, COBOL/370, or IBM COBOL that are invoked by assembler programs must be entered in the same AMODE each time they are called.

To avoid the abend in the example above:

- In Step 2, `AssemblerA` must save the AMODE.
- In Step 3, `AssemblerA` needs to BASSM to `COBOLB` instead of BALR.
- In Step 4, `AssemblerA` need to restore the AMODE saved in Step 2.

Note: If an assembler program that is AMODE 31 calls a COBOL program that is AMODE 24, the assembler program must also be RMODE 24 in order for COBOL to return to the assembler program. If the assembler program is AMODE ANY in this case, an abend might result upon return from the COBOL program as a result of branching to an invalid address. This is because R14 will contain a 31-bit address from the assembler program's save area, but COBOL will return to the assembler program in AMODE 24.

Appendix E. Debugging tool comparison

The Debug Tool is a program analyzer that runs within Language Environment and supports a number of high-level languages, including IBM COBOL.

For COBOL for MVS & VM and COBOL for OS/390 & VM, the Debug Tool, is orderable as a feature of the compiler.

Debugging existing applications

The Debug Tool does not support debugging of OS/VS COBOL programs.

The Debug Tool provides support to VS COBOL II **Release 3.0 and higher** if the VS COBOL II programs have been compiled with the TEST compiler option. Considerations for debugging VS COBOL II programs include:

COBTEST
COBTEST can still be used to debug VS COBOL II programs that use the VS COBOL II library. COBTEST is incompatible with the Debug Tool, and cannot be used with Language Environment.

Test scripts
If you use COBTEST to run program tests and have a library of COBTEST test scripts, the Debug Tool contains a command translator to aid in your conversion efforts.

Debugging migrated applications

When you compile your OS/VS COBOL or VS COBOL II programs with IBM COBOL, you need to know the following:

Applications with OS/VS COBOL programs

The Debug Tool can be used to debug a run unit that contains a combination of IBM COBOL and OS/VS COBOL programs, but no information about the OS/VS COBOL programs is available to the Debug Tool.

If a run unit contains OS/VS COBOL programs that specify the FLOW or COUNT compiler option, you cannot use the Debug Tool to debug **any** programs in the run unit (including VS COBOL II and IBM COBOL programs).

Applications with VS COBOL II programs

FDUMP compiler option
The FDUMP compiler option is not supported in IBM COBOL, but its function is. FDUMP is mapped to TEST(SYM) compiler option.

TEST compiler option
The TEST compiler option syntax has been modified to allow you to specify if and where compiled-in hooks should be located, and if dictionary and CALC tables should be generated. Five hook-location suboptions are possible: ALL, NONE, STMT, PATH, and BLOCK; two symbol-table suboptions are possible: SYM and NOSYM.

In an application that mixes VS COBOL II programs and IBM COBOL programs, the Debug Tool can only debug the VS COBOL II programs that are compiled with TEST.

Batch debugging considerations

The Debug Tool provides a function equivalent to the batch mode of COBTEST, along with some additional features.

In batch mode, the Debug Tool gets its input from an input file and the Debug Tool output is written to an output file similar to COBTEST batch mode. The differences between the Debug Tool batch mode and COBTEST batch mode are that the COBTEST commands, RECORD, QUALIFY, RESTART, GO, and RUN, behave differently in batch mode than in interactive mode. With the Debug Tool, all commands behave the same in both modes.

Some Debug Tool commands, such as fullscreen commands, are not allowed in batch mode.

Initiating the Debug Tool

The way in which you initiate the debugging tool is different for IBM COBOL programs. When using the Debug Tool, the application program starts first and the Language Environment run-time TEST option controls the invocation of the Debug Tool. (With VS COBOL II, you are required to invoke COBTEST first; it then starts the application to be debugged.)

You can also invoke the Debug Tool directly from your application by using the Language Environment callable service CEETEST. A brief description of these two methods follow.

TEST run-time option

The Language Environment TEST run-time option is used to determine if the Debug Tool is to be invoked when an application program is run with Language Environment. Invocation can be immediate or deferred, depending on the option subparameters.

The IBM-supplied default is NOTEST. This specifies that the Debug Tool is not to be initialized to process the initial command string nor is it to be initialized for any program condition that might arise when running the program. However, if debugging services are needed, you can invoke the Debug Tool using the library service CEETEST.

For detailed information on the Language Environment TEST option subparameters and suboptions, see *Language Environment Programming Reference*.

CEETEST

Language Environment provides callable service CEETEST to allow the Debug Tool to gain control, and to specify a string of commands to be passed to the Debug Tool. Calling this service, causes the Debug Tool to be initialized and invoked. (If the Debug Tool is already initialized, then this re-entry is similar to a breakpoint.)

When using CEETEST to invoke the Debug Tool, the string parameter containing a command list is optional. If you do use a command list, the commands are passed to the Debug Tool and executed. If the command list does not contain any GO, GOTO, STEP, or QUIT commands, commands will then be requested from the terminal or the primary commands file. If the GO

command is encountered at any point (command list, terminal, or commands file), the Debug Tool returns to the application program at the point following the service call and your program continues running.

For detailed information and examples of the Language Environment callable service CEETEST, see *Language Environment Programming Reference*.

Command language comparison

Figure 55 compares the command language of TESTCOB, COBTEST, and the Debug Tool. As you can see, many of the Debug Tool commands are different from the commands used in COBTEST and TESTCOB. In most cases, the function is identical.

Commands not supported by a specific debugger are indicated with a dash.

Figure 55 (Page 1 of 3). Debugger command language comparison

TESTCOB command	COBTEST command	Debug Tool command	Function
AT	AT	AT	Set a breakpoint
–	AUTO[2]	MONITOR LIST	Automatically monitor variables
–	COLOR[2]	PANEL COLORS	Display panel to set color attributes
–	DOWN[9]	WINDOW DOWN	Move window down
DROP	DROP	CLEAR	Delete a defined symbol
DUMP	DUMP	CALL %DUMP	Produce a memory dump
END	QUIT	QUIT	End the debug session
EQUATE	EQUATE	SET EQUATE or CLEAR EQUATE	Define a symbol
–	FLOW	LIST LAST	Collect control flow information
–	FREQ	SET FREQUENCY	Tally the execution counts for verbs
GO	GO	GO	Start execution of the COBOL program
HELP (TSO only)	HELP[3]	HELP	Provide online help information
IF	IF	IF	Evaluate a condition
–	LEFT[9]	WINDOW LEFT	Move window left
–	LINK	LOAD or CALL[5]	Set up a LINKAGE SECTION
LIST	LIST	LIST, DESCRIBE[6]	List contents of variables
LISTBRKS	LISTBRKS	LIST AT	List breakpoints in effect
–	LISTEQ	QUERY EQUATES	List the defined symbols
LISTFILE	LIST	DESCRIBE ATTRIBUTES	List attributes of a file
–	LISTFREQ	LIST FREQUENCY	List frequency counts of verbs
–	LISTINGS[2]	PANEL LISTINGS	Display panel associating program with a listing
–	MOVECURS[2]	CURSOR	Move cursor from command line to a window or vice versa
NEXT	NEXT	STEP	Set temporary breakpoint at next verb
–	NORECORD	SET LOG OFF	Turn off debug session logging

Figure 55 (Page 2 of 3). Debugger command language comparison

TESTCOB command	COBTEST command	Debug Tool command	Function
OFF	OFF	CLEAR AT	Turn off a breakpoint in effect
OFFWN	OFFWN	CLEAR AT	Turn off a conditional breakpoint
–	ONABEND	AT OCCURRENCE CEExxxx	Perform an action at occurrence of a Language Environment condition, where xxxx is the condition returned from Language Environment
–	PEEK	QUERY prefix	Show breakpoints within a line
–	POSITION[2]	SCROLL TO	Move to a certain location of a displayed object
–	PREVDISP[2]	–[7]	Show the last displayed user ISPF screen
–	PRINTDD	–[1]	Route the output to a data set
–	PROC	AT CALL	Trap CALLS to certain subprograms
–	PROFILE[2]	PANEL PROFILE	Display panel to set user profile attributes
QUALIFY	QUALIFY	SET QUALIFY	Change the current program qualify
–	RECORD	SET LOG ON	Turn on logging of debug session
–	RESTART	RESTART[8]	Reinitialize program without exiting debugger
–	RESTORE[2]	SET or QUALIFY RESET	Return to current point of execution
–	RIGHT[9]	WINDOW RIGHT	Move window right
RUN	RUN	CLEAR AT then GO	Remove breakpoints and run to completion
–	SEARCH[2]	FIND	Search for a string in displayed object
–	SELECT	SHOW	Display a specific frequency count
SET	SET	MOVE or SET or COMPUTE	Alter contents of a variable
SOURCE	SOURCE[4]	WINDOW OPEN	Display source program statements
–	STEP	STEP	Execute specified number of verbs
–	SYSCMD	SYSTEM	Places you in CMS subset mode
–	SUFFIX[2]	SET SUFFIX	Turn on visual display of frequency
TRACE	TRACE	AT GLOBAL STATEMENT or AT GLOBAL PATH	Trace flow of execution
–	UP[9]	WINDOW UP	Move window up
–	VTRACE[2]	STEP	Dynamic visual trace of program
WHEN	WHEN	AT * IF...	Set up conditional breakpoint

Figure 55 (Page 3 of 3). Debugger command language comparison

TESTCOB command	COBTEST command	Debug Tool command	Function
–	WHERE	QUERY LOCATION	List current point of execution

Notes to debugger command language comparison:

1 The PRINTDD command is unnecessary for the Debug Tool. The print file has a fixed ddname (although, the ddname can be specified at invocation).

2 These commands are supported only with COBTEST FULL SCREEN debugging.

3 HELP is only supported under TSO and CMS, with COBTEST LINE MODE debugging.

4 The COBTEST SOURCE command is only supported in FULL SCREEN debugging. Also, the connotation is different from the TESTCOB SOURCE command.

5 LINK in COBTEST provides real storage for the data items in the LINKAGE SECTION. The Debug Tool accomplishes this by declaring those data names. It then calls the COBOL program, passing the necessary parameters.

6 The Debug Tool does not have a facility to list a range of names. However, it is possible to list all names for a block, or all names matching a specific pattern.

7 The PREVDISP command is no longer required. The Debug Tool does not use ISPF for display.

8 The RESTART command is available only with the programmable workstation. Note, the RESTART command does not retain break settings as it did in COBTEST.

9 The DOWN, LEFT, RIGHT, and UP commands are executed by ISPF. They are not actual COBTEST commands.

Appendix F. Compiler option comparison

This appendix describes the IBM COBOL compiler options. Figure 56 shows how the IBM COBOL compiler options compare with those of VS COBOL II and OS/VS COBOL. For complete descriptions of the IBM COBOL options, see the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.

Figure 56 (Page 1 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
ADATA			√	√	√	Produces associated data file at compilation. NOADATA is the default. The IBM COBOL ADATA option replaces the COBOL/370 EVENTS option.
ANALYZE			√	√	√	Causes the compiler to check the syntax of embedded SQL and CICs statements in addition to native COBOL statements.
ADV	√	√	√	√	√	Adds print control byte at beginning of records. ADV is the default.
ALOWCBL		√	√	√	√	Allows PROCESS or CBL statements in source program. You can only specified only at installation time. ALOWCBL is the default.
APOST	√	√	√	√	√	Specifies apostrophe (') as delimiter for literals. QUOTE is the default. In IBM COBOL, literals can be delimited with either quotes or apostrophes regardless of whether APOST or QUOTES is in effect. If APOST is used, the figurative constant QUOTE/QUOTES represents one or more apostrophe (') characters.
ARITH					√	Sets the maximum number of digits that you can specify for decimal data and affects the precision of intermediate results. With ARITH(COMPAT) you can specify 18 digits in the PICTURE clause, fixed-point numeric literals, and arguments to NUMVAL and NUMVAL-C, and 28 digits in arguments to FACTORIAL. With ARITH(EXTEND) you can specify 31 digits in the PICTURE clause, fixed-point numeric literals, and arguments to NUMVAL and NUMVAL-C, and 29 digits in arguments to FACTORIAL.
AWO		√	√	√	√	Activates APPLY WRITE-ONLY processing for physical sequential files with VB format. NOAWO is the default.
BUF	√					Allocates buffer storage for compiler work data sets. In IBM COBOL, the BUFSIZE option replaces the OS/VS COBOL BUF options.

Figure 56 (Page 2 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
BUFSIZE		√	√	√	√	Allocates buffer storage for compiler work data sets. Three suboptions are available: BUFSIZE(nnnnn), BUFSIZE(nnnK), and BUFSIZE(4096). BUFSIZE(4096) is the default. BUFSIZE replaces the OS/VS COBOL BUF option.
CLIST	√					Produces condensed PROCEDURE DIVISION listing plus tables and program statistics. NOCLIST is the default. The VS COBOL II and IBM COBOL OFFSET option replaces the OS/VS COBOL CLIST option.
CMPR2		√	√	√	√	Specifies generation of IBM COBOL source code compatible with VS COBOL II Release 2 or other VS COBOL II CMPR2 behavior. NOCMPR2 is the default. NOCMPR2 specifies the full use of all IBM COBOL language features (including language extensions for object-oriented COBOL and improved interoperability with C programs).
COMPILE		√	√	√	√	Requests an unconditional full compilation. Other options are NOCOMPILE and NOCOMPILE(W|E|S). The default is NOCOMPILE(S). NOCOMPILE specifies unconditional syntax checking. NOCOMPILE(W|E|S) specify conditional syntax checking based on the severity of the error. COMPILE is equivalent to the OS/VS COBOL NOSYNTAX and NOCSYNTAX options. NOCOMPILE is equivalent to the OS/VS COBOL SYNTAX options. NOCOMPILE(W|E|S) is equivalent to the OS/VS COBOL CSYNTAX and SUPMAP options.
CURRENCY			√	√	√	Defines default currency symbol. When both the CURRENCY option and the CURRENCY SIGN clause are used in a program, the symbol specified in the CURRENCY SIGN clause is considered the currency symbol in a PICTURE clause when that symbol is used. NOCURRENCY is the default and indicates that no alternate default currency sign is provided by the CURRENCY option.
DATA(24) DATA(31)		√	√	√	√	Specifies whether reentrant program data areas reside above or below the 16M line. With DATA(24) reentrant programs must reside below the 16m line. With DATA(31) reentrant programs can reside above the 16M line. DATA(31) is the default.

Figure 56 (Page 3 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
DATEPROC			√	√	√	Enables the millennium language extensions of the COBOL compiler. Options consist of DATEPROC(FLAG), DATEPROC(NOFLAG), DATEPROC(TRIG), DATEPROC(NOTRIG) and NODATEPROC.
DBCS		√	√	√	√	Tells compiler to recognize DBCS shift-in and shift-out codes. NODBCS is the default, and tells compiler to treat DBCS shift-in and shift-out codes as ordinary data characters.
DBCSXREF=code		√	√	√	√	Specifies that an ordering program is to be used for cross-references to DBCS characters, where **code** sets parameters giving information about the DBCS Ordering Support Program. You can only specify DBCSXREF at installation time. DBCSXREF=NO is the default.
DECK	√	√	√	√	√	Generates object code as 80-character card images and places it in SYSPUNCH file. NODECK is the default.
DIAGTRUNC					√	Causes the compiler to issue a severity-4 (warning) diagnostic for MOVE statements with numeric receivers when the receiving data has fewer integer positions than the sending data item or literal.
DLL				√	√	Enables the compiler to generate an object module that is enabled for Dynamic Link Library (DLL) support. NODLL is the default.
DMAP	√					Produces listing of Data Division and implicitly declared items. NODMAP is the default. The VS COBOL II and IBM COBOL MAP option replaces the OS/VS COBOL DMAP option.
DUMP	√	√	√	√	√	Specifies that a system dump be produced at end of compilation. NODUMP is the default.
DYNAM	√	√	√	√	√	Changes the behavior of CALL literal statements to load subprograms dynamically at run time. NODYNAM is the default. With NODYNAM, CALL literal statements cause subprograms to be statically link-edited in the load module.
EXIT(IN-id) EXIT(LIB-id) EXIT(PRT-id) EXIT(ADT-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.) The ADT-id suboption is only available with COBOL for MVS & VM and COBOL for OS/390 & VM. NOEXIT is the default.

Figure 56 (Page 4 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
EXPORTALL				√	√	Instructs the compiler to automatically export certain symbols when the object deck is link-edited to form a DLL. NOEXPORTALL is the default. EXPORTALL is not applicable under VM/CMS.
FASTSRT		√	√	√	√	Specifies fast sorting by the IBM DFSORT licensed program. NOFASTSRT is the default, and specifies that IBM COBOL will do sort/merge input/output.
FLAG	√	√	√	√	√	Specifies that syntax messages are produced at the level indicated. For OS/VS COBOL the FLAG options are: FLAGW and FLAGE. For IBM COBOL, the FLAG options are: FLAG(I) FLAG(W) FLAG(E) FLAG(S) FLAG(U) FLAG(I|W|E|S|U,I|W|E|S|U) For VS COBOL II and IBM COBOL, FLAG(I) is the default.
FLAGMIG		√	√	√	√	Specifies IBM COBOL NOCMPR2 flagging for possible semantic changes from VS COBOL II Release 2 or other programs with CMPR2 behavior. NOFLAGMIG is the default.
FLAGSTD		√	√	√	√	Specifies COBOL 85 Standard flagging. For COBOL for OS/390 & VM and COBOL for MVS & VM FLAGSTD also flags language syntax for object-oriented COBOL, improved interoperability, and use of the PGMNAME(LONGMIXED) compiler option. NOFLAGSTD is the default.
FDUMP		√				Use to produce a dump with debugging information when an application ends with an abend. NOFDUMP is the default. The IBM COBOL TEST(SYM) option replaces the VS COBOL II FDUMP option.
IDLGEN			√	√	√	In addition to the normal compile of the COBOL source file, IDLGEN generates IDL definitions for defined classes. NOIDLGEN is the default.
INTDATE(ANSI) INTDATE(LILIAN)			√	√	√	Determines the starting date for integer format dates when used with date intrinsic functions. ANSI uses the COBOL 85 Standard starting date, where Day 1 = January 1, 1601. LILIAN uses the Language Environment Lilian starting date, where Day 1 = October 15, 1582. INTDATE(ANSI) is the default.

Figure 56 (Page 5 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
LANGUAGE		√	√	√	√	LANGUAGE(AAa...a) specifies language in which compiler messages are issued, where AAa...a is: **UE or UENGLISH** Uppercase English **EN or ENGLISH** Mixed-case English **JA, JP, or JAPANESE** Japanese, using the KANJI character set LANGUAGE=(EN) is the default.
LIB	√	√	√	√	√	Specifies that the program uses the COPY library. The default is NOLIB.
LINECNT=nn	√					Specifies the number of lines per page on the output listing. For VS COBOL II and IBM COBOL, the LINECOUNT compiler option replaces the OS/VS COBOL LINECNT option.
LINECOUNT		√	√	√	√	Specifies the number of lines per page on the output listing. The two formats for LINECOUNT are: LINECOUNT(60) and LINECOUNT(nn). LINECOUNT(60) is the default. LINECOUNT replaces the OS/VS COBOL LINECNT option.
LIST		√	√	√	√	Produces listing of assembler language expansion of source code. NOLIST is the default. LIST replace the OS/VS COBOL PMAP option.
LOAD	√					Stores object code on disk or tape for input to linkage editor. NOLOAD is default. The VS COBOL II and IBM COBOL OBJECT option replaces the OS/VS COBOL LOAD option.
MAP		√	√	√	√	Produces listing of Data Division and implicitly declared items NOMAP is the default. MAP replaces the OS/VS COBOL DMAP option.
NAME	√	√	√	√	√	Indicates that a linkage editor NAME statement is appended to each object module created. For VS COBOL II and IBM COBOL, NAME has the sub-options (ALIAS|NOALIAS). If ALIAS is specified, an ALIAS statement is also generated for each ENTRY statement NONAME is the default.
NUM	√					Prints line numbers in error messages and listings. NONUM is the default. The VS COBOL II and IBM COBOL NUMBER option replaces the OS/VS COBOL NUM option.

Figure 56 (Page 6 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
NUMBER		√	√	√	√	Prints line numbers in error messages and listings. NONUMBER is the default. The NUMBER option replaces the OS/VS COBOL NUM option.
NUMCLS		√	√	√	√	Determines, together with the NUMPROC option, valid sign configurations for numeric items in the NUMERIC class test. NUMCLS has two sub-options: (PRIM/ALT). NUMCLS(PRIM) is the default. You can specify NUMCLS only at installation time. For more information, see the: • Program Directory for VM • *Installation and Customization under OS/390* for COBOL for OS/390 & VM • *Installation and Customization under MVS* for COBOL for MVS & VM
NUMPROC		√	√	√	√	Handles 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)** IBM COBOL processes sign conversion in a manner very similar to OS/VS COBOL NUMPROC(NOPFD) is the default.
OBJECT		√	√	√	√	Stores object code on disk or tape for input to linkage editor. NOOBJECT is the default. OBJECT replaces the OS/VS COBOL LOAD option.
OFFSET		√	√	√	√	Produces condensed PROCEDURE DIVISION listing plus tables and program statistics. NOOFFSET is the default. OFFSET replaces the OS/VS COBOL CLIST option.
OPTIMIZE	√	√	√	√	√	Optimizes object program. With IBM COBOL, OPTIMIZE has the suboptions of (STD/FULL). OPTIMIZE(FULL) provides improved run-time performance, over both the OS/VS COBOL and VS COBOL II OPTIMIZE option, because the compiler discards unused data items and does not generate code for any VALUE clauses for these data items. NOOPTIMIZE is the default.

Figure 56 (Page 7 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
OUTDD(SYSOUT) OUTDD(ddname)		√	√	√	√	Routes DISPLAY output to SYSOUT or to a specified data set. OUTDD(SYSOUT) is the default. OUTDD replaces the OS/VS COBOL SYSx option.
PGMNAME			√	√	√	Controls the handling of program names in relation to length and case. **PGMNAME(LONGMIXED)** Program names are used at their full length, without truncation and without folding or translating by the compiler. **PGMNAME(LONGUPPER)** Program names are used at their full length, without truncation. **PGMNAME(COMPAT)** Program names are processed similarly to Release 1. PGMNAME(COMPAT) is the default.
PMAP	√					Produces listing of assembler language expansion of source code. NOPMAP is the default. The VS COBOL II and IBM COBOL LIST compiler option replaces the OS/VS COBOL PMAP option.
QUOTE	√	√	√	√	√	Specifies quotation mark (") as delimiter for literals. QUOTE is the default. In IBM COBOL, literals can be delimited with either quotes or apostrophes regardless of whether APOST or QUOTES is in effect. If QUOTE is used, the figurative constant QUOTE/QUOTES represents one or more quotation marks (") characters.
RES	√	√				Causes most library routines to be loaded dynamically, instead of being link-edited with the COBOL program.
RENT		√	√	√	√	Specifies reentrant code in object program. NORENT is the default.
RMODE(AUTO) RMODE(24) RMODE(ANY)			√	√	√	Allows NORENT programs to have RMODE(ANY). RMODE(AUTO) is the default.
SEQ	√					Checks ascending sequencing of source statement line numbers. SEQ is the default. The VS COBOL II and IBM COBOL SEQUENCE option replaces the OS/VS COBOL SEQ option.
SEQUENCE		√	√	√	√	Checks ascending sequencing of source statement line numbers. SEQUENCE is the default. SEQUENCE replaces the OS/VS COBOL SEQ option.

Figure 56 (Page 8 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
SIZE(MAX) SIZE(nnnnn) SIZE(nnnK)		√	√	√	√	Specifies virtual storage to be used for compilation. SIZE(MAX) is the default.
SOURCE	√	√	√	√	√	Produces a listing of the source program and embedded messages. SOURCE is the default.
SPACE	√	√	√	√	√	Produces a single, double, or triple spaced listing. The syntax of the SPACE option in OS/VS COBOL is: SPACE1, SPACE2, SPACE3. The syntax of SPACE in VS COBOL II and IBM COBOL is: SPACE(1), SPACE(2), SPACE(3). SPACE(1) is the default.
SQL					√	Enables the DB2 coprocessor capability and specifies DB2 suboptions.
SSRANGE		√	√	√	√	At run time, checks validity of subscript, index, and reference modification references NOSSRANGE is the default.
SYSx	√					Routes DISPLAY output to SYSOUT or to a specified data set. The VS COBOL II and IBM COBOL OUTDD option replaces the OS/VS COBOL SYSx option.
STATE	√					Use to produce a dump with debugging information when an application ends with an abend. NOSTATE is the default. The IBM COBOL TEST(NONE,SYM) option replaces the OS/VS COBOL STATE option.
SUPMAP SYNTAX CSYNTAX	√					Specifies the extent of compilation. SYNTAX specifies unconditional syntax checking. CSYNTAX and CSUPMAP specify conditional syntax checking. NOSYNTAX and NOCSYNTAX specify an unconditional full compile. The VS COBOL II and IBM COBOL COMPILE option replaces the OS/VS COBOL SYNTAX, CSYNTAX, and CSUPMAP options.
SXREF	√					Produces sorted cross-reference listing of data names and procedure names used in program. The default is NOSXREF. The VS COBOL II and IBM COBOL XREF option replaces the OS/VS COBOL SXREF option.
TERM	√					Sends progress messages to the SYSTERM data set. NOTERM is the default. The VS COBOL II and IBM COBOL TERMINAL option replaces the OS/VS COBOL TERM option.

Figure 56 (Page 9 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
TERMINAL		√	√	√	√	Sends progress messages to the SYSTERM data set. NOTERMINAL is the default. TERMINAL replaces the OS/VS COBOL TERM option.
TEST	√	√	√	√	√	Produces object code usable by the debug tool for the product. NOTEST is the default. For details on the suboptions for the IBM COBOL TEST option, see the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.
TRUNC	√	√	√	√	√	Truncates final intermediate results. OS/VS COBOL has the TRUNC and NOTRUNC options (NOTRUNC is the default). VS COBOL II and IBM COBOL have the TRUNC(STD|OPT|BIN) option. **TRUNC(STD)** Truncates numeric fields according to PICTURE specification of the binary receiving field **TRUNC(OPT)** Truncates numeric fields in the most optimal way **TRUNC(BIN)** Truncates binary fields based on the storage they occupy TRUNC(STD) is the default. For a complete description, see the *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.
TYPECHK			√	√	√	Enforces the rules for OO type conformance and issues diagnostics for any violations. The default is NOTYPECHK.
VBREF VBSUM	√	√	√	√	√	Produces cross-reference listing of all verb types used in program. Only OS/VS COBOL supports VBSUM. The default is NOVBREF (as well as NOVBSUM for OS/VS COBOL).
WORD		√	√	√	√	Tells compiler which reserved word table to use. To use an installation-specific reserved word table, specify WORD(table-name). To use the default reserved word table, specify NOWORD. NOWORD is the default.

Figure 56 (Page 10 of 10). Compiler option comparison

Option	Available in					Usage notes
	OSVS	VSII	MVS VM	390 V2R1	390 V2R2	
XREF		√	√	√	√	Produces sorted cross-reference listing of data names and procedure names used in program. The default is NOXREF. XREF replaces the OS/VS COBOL SXREF option.
YEARWINDOW			√	√	√	Specifies the first year of the 100-year window (the century window) to be applied to windowed date field processing by the COBOL compiler.
ZWB	√	√	√	√	√	Removes sign from a signed numeric DISPLAY field when comparing it with an alphanumeric field. ZWB is the default.
Options for CMS only						
DISK	√	√	√	√	√	Directs program listings to a disk LISTING file.
PRINT	√	√	√	√	√	Directs program LISTING file to the virtual printer
NOPRINT	√	√	√	√	√	Specifies that no LISTING file be produced

Appendix G. Compiler limit comparison

The following table lists the compiler limits for COBOL for MVS & VM, VS COBOL II, and OS/VS COBOL programs running under MVS and COBOL for OS/390 & VM program running under OS/390, Other operating systems might impose further limits.

The numbers are **guidelines** to the limits.

- Interpret a limit stated in megabytes (M) as: x megabytes minus 1 byte.
- Interpret a limit stated in kilobytes (K) as: x kilobytes minus 1 byte.
- Interpret a limit stated in gigabytes (G) as: x gigabytes minus 1 byte.
- B stands for bytes.
- N/L stands for no limit.
- Footnotes are at the end of the table.

Language element	IBM COBOL	VS COBOL II	OS/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/L
COPY REPLACING ... BY ... (items per COPY statement)	N/L	N/L	150
Number of COPY libraries	N/L	N/L	N/L
Block size of COPY library	32,767B	32,767B	16K
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/L	N/L	N/L
literal THRU/ALSO ...	256	256	256
Input-Output Section			
FILE-CONTROL paragraph			
SELECT file-name ...	A maximum of 65,535 file names can be assigned external names	A maximum of 65,535 file names can be assigned external names	A maximum of 64K file names can be assigned external names
ASSIGN system-name ...	N/L	N/L	N/L
ALTERNATE RECORD KEY data-name ...	253	253	253
RECORD KEY length	N/L[3]	N/L[3]	255
RESERVE integer (buffers)	255[4]	255[4]	255[4]
I-O-CONTROL paragraph			

Language element	IBM COBOL	VS COBOL II	OS/VS COBOL
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/L[2]	N/L[2]	N/L[2]
MULTIPLE FILE ... file-name	N/L[2]	N/L[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/L[2]	N/L[2]	N/L[2]
BLOCK CONTAINS integer	2,147,483,647[9]	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/L[2]	N/L[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

Language element	IBM COBOL	VS COBOL II	OS/VS COBOL
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/L	N/L	N/L
VALUE literal ...	N/L	N/L	N/L
66 RENAMES ...	N/L	N/L	N/L
PICTURE character-string	30	30	30
Numeric item digit positions	If the	18	18
Num-edit character positions	ARITH(COMPAT)		
PICTURE replication ( )	compiler option		
PIC repl (editing)	is in		
DBCS Picture Reactivation	affect: 18		
Group item size:			
FILE SECTION	If the		
Elementary item size	ARITH(EXTEND)		
VALUE initialization	compiler option		
(Total length of value literals)	is in		
OCCURS integer	affect: 31		
Total number of ODOs	249	249	127
Levels of ODO	16,777,215	16,777,215	99999
Table size	32,767	32,767	99999
Table element	8,388,607		
ASCENDING/DESCENDING KEY ...			
(per OCCURS clause)	1,048,575		
Total length	16,777,215	16,777,215	32K
INDEXED BY ... (index names)	16,777,215	16,777,215	64K
Total num of indexes (index names)			
Size of relative index	16,777,215	16,777,215	32K
	4,194,303[1]	4,194,303[1]	64K[1]
	N/L	N/L	3
	16,777,215	16,777,215	32K
	8,388,607	8,388,607	32K
	12	12	12
	256B	256B	256B
	12	12	12
	65,535	65,535	64K
	32,765	32,765	32K
Linkage Section	134,217,727	134,217,727	1M
Total 01 + 77 (data items)	N/L	N/L	255
PROCEDURE DIVISION			

Language element	IBM COBOL	VS COBOL II	OS/VS COBOL
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/L	N/L	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/L
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/L	N/L	64K
CANCEL id/lit ...	N/L	N/L	N/L
CLOSE file-name ...	N/L	N/L	N/L
COMPUTE identifier ...	N/L	N/L	N/L
DISPLAY id/lit ...	N/L[8]	N/L[8]	N/L[8]
DIVIDE identifier ...	N/L	N/L	N/L
ENTRY USING id/lit ...	N/L	N/L	N/L
EVALUATE ... subjects	64	64	N/L
EVALUATE ... WHEN clauses	256	256	N/L
GO pn ... DEPENDING	255	255	2031
INSPECT TALLYING/REPLACING	N/L	N/L	15
MERGE file-name			
ASCENDING/DESCENDING KEY ...	N/L	N/L	12
Total key length	4092B[6]	4092B[6]	256
USING file-name ...	16[7]	16[7]	16[7]
MOVE id/lit TO id ...	N/L	N/L	N/L
MULTIPLY identifier ...	N/L	N/L	N/L
OPEN file-name	N/L	N/L	N/L
PERFORM	4,194,303	4,194,303	64K
SEARCH ... WHEN ...	N/L	N/L	N/L
SEARCH ALL ... WHEN ...	12	12	12
SET index/id ... TO	N/L	N/L	N/L
SET index ... UP/DOWN	N/L	N/L	N/L
SORT file-name			
ASCENDING/DESCENDING KEY	N/L	N/L	12
Total key length	4092B[6]	4092B[6]	256
USING file-name ...	16[7]	16[7]	16[7]
STRING identifier ...	N/L	N/L	N/L
DELIMITED id/lit ...	N/L	N/L	N/L
UNSTRING id			
DELIMITED id/lit OR id/lit ...	255	255	15
INTO id/lit ...	N/L	N/L	N/L
USE ... ON file-name ...	N/L	N/L	N/L

Notes to compiler limits table:

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 QSAM limit.

5 Compiler limit shown, but QSAM limits it to 32,767 bytes.

6 Limit is 4088 bytes if EQUALS is coded on the OPTION control statement.

[7] SORT limit.

[8] The compiler limit is shown; however, Language Environment limits the maximum length of a data item you can display with DISPLAY UPON SYSOUT to 16,384.

[9] Requires large block interface (LBI) support provided by OS/390 DFSMS Version 2 Release 10.0 or later. On OS/390 systems with earlier releases of DFSMS, the limit is 32,767 bytes. For more information on using large block sizes, see the *IBM COBOL for OS/390 & VM Programming Guide*.

For additional information on using DISPLAY with OS/VS COBOL programs, see "Understanding SYSOUT output changes" on page 68.

For additional information on using DISPLAY with VS COBOL II programs, see "Understanding SYSOUT output changes" on page 88.

Appendix H. Industry standards

IBM COBOL supports the following industry standards:

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.

   All required modules are supported at the highest level defined by the standard. In the following list, the shorthand notation for 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). The first digit indicates the level of language elements within the module supported by IBM COBOL. Next is the three-character abbreviation of the module name as used in the standard. Finally, the two digits separated by a comma indicate the minimum and maximum levels of the module. For example, (2 SEQ 1,2) means that IBM COBOL supports the sequential I-O module at level 2, while the range of levels in the module is from 1 (minimum) to 2 (maximum).

   - Nucleus (2 NUC 1,2)

     Provides internal processing of data within the four basic divisions of a program and the capability for defining and accessing tables.

   - Sequential I-O (2 SEQ 1,2)

     Provides access to records of a file in established sequence. The sequence is established as a result of writing the records to the file.

   - Relative I-O (2 REL 0,2)

     Provides access to records in either a random or sequential manner. Each record is uniquely identified by an integer specifying the record's logical position in a file.

   - Indexed I-O (2 INX 0,2)

     Provides access to records in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of a key within that record.

   - Sort-Merge (1 SRT 0,1)

     Orders one or more files of records, or combines two or more identically ordered files of records, according to a set of user-specified keys.

   - Inter-Program Communication (2 IPC 1,2)

Allows a COBOL program to communicate with other programs through transfers of control and access to common data items.

- Source Text Manipulation (2 STM 0,2)

  Allows the insertion of source program text as part of the compilation of the source program. COBOL libraries contain texts which are available to the compiler at compile time and which can be treated by the compiler as part of the source program.

In addition, the following optional modules of the standard are supported:

- Intrinsic Functions (1 ITR 0,1)

  Provides the capability to reference a data item whose value is derived automatically at the time of reference during the execution of the object program.

- Debug (1 DEB 0,2)

  Monitors object program execution through declarative procedures, special debugging lines, and a special register, DEBUG-ITEM, which gives specific information about execution status.

- Segmentation (2 SEG 0,2)

  Refreshes independent segments when required.

The following optional modules of the standard are not supported:

- Report Writer
- Communications
- Debug (2 DEB 0,2)

3. FIPS Publication 21-4, Federal Information Processing Standard 21-4, 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*.

Note: The term "COBOL 85 Standard" is used in this book to refer to the combination of the following standards:

1. ISO 1989:1985, Programming languages - COBOL.

   ISO 1989/Amendment 1, Programming languages - COBOL - Amendment 1: Intrinsic function module.

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.

Note: The term "COBOL 74 Standard" is used in this book to refer to X3.23-1974, American National Standard for Information Systems - Programming Language - COBOL.

Appendix I. Preventing file status 39 for QSAM files

This appendix provides guidelines to help prevent common File Status 39 problems for QSAM files, which are due to mismatches in the attributes for file organization, record format (fixed or variable), record length, or the code set.

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 data set, 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 data set.
For Format-F files	The record length specified in your program must exactly match the length attribute of the data set.
For Format-U files	The maximum record length specified in your program must exactly match the length attribute of the data set.

Note: Remember that information in the JCL overrides information in the data set label.

For details on how record lengths are determined from the FD entry and record descriptions in your program, see *IBM COBOL Programming Guide* for the version of IBM COBOL you are using.

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 data set 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 data set. 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.

If the existing file is an ASCII data set (DCB=(OPTCD=Q)), you must specify the CODE-SET clause in the program's FD entry for the file.

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 DD 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 DD statement related to the FILE-CONTROL and FD entries in your program:

```
JCL DD Statement:

     1
     |
     v
//OUTFILE  DD  DSNAME=OUT171,UNIT=SYSDA,SPACE=(TRK,(50,5)),
//             DCB=(BLKSIZE=400)


/*


IBM COBOL Program Code:

ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
    FILE-CONTROL.

      SELECT CARPOOL <---- 2
          ASSIGN TO OUTFILE <---- 1
          ORGANIZATION IS SEQUENTIAL
          ACCESS IS SEQUENTIAL.
    .
    .
    .
DATA DIVISION.
  FILE SECTION.

    FD CARPOOL <---- 2
      LABEL RECORD STANDARD
      BLOCK CONTAINS 0 CHARACTERS
      RECORD CONTAINS 80 CHARACTERS
```

Figure 57. Example of JCL, FILE-CONTROL entry, and FD entry

Where:

1 The *ddname* in the DD statement corresponds to the *assignment-name* in the ASSIGN clause:

```
//OUTFILE  DD  DSNAME=OUT171 ...
```

This *assignment-name* points to the *ddname* of OUTFILE in the DD 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 data set, (for example, you are using an ISPF allocation panel or if your DD statement is for a batch job in which the program uses RECORD CONTAINS 0):

- 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 in your DD statement or allocation should be 54.

Processing files dynamically created by COBOL: When a file is not made available with a DD statement or a TSO ALLOCATE command and your COBOL program defines that the file be created, IBM COBOL will dynamically allocate the file. When the file is opened, the file attributes specified in your program will be used; no file attributes conflicts will exist.

Appendix J. Overriding linkage editor defaults

This appendix gives you the instructions necessary to override the default AMODE and RMODE settings assigned by the IBM COBOL compiler based on the use of the RENT and RMODE compiler options.

When you should not override the default settings: Do not override the default AMODE/RMODE settings in the following cases:

AMODE

For load modules that contain VS COBOL II programs compiled NORES or any OS/VS COBOL programs, **do not** specify a linkage editor override of AMODE ANY or AMODE 31. The only exception is if the programs are external entry points called by the system or through system services and the logic of the application can guarantee, through appropriate AMODE switching, that these programs will be entered in AMODE 24. These programs will not switch AMODEs when they statically call other programs.

RMODE

For load modules that contain VS COBOL II programs compiled NORES and NORENT or any OS/VS COBOL programs, **do not** specify a linkage editor override of RMODE ANY. This is because certain control blocks contained in the object modules produced by the compiler must reside below the 16M line.

When you should override the default settings: For load modules with both IBM COBOL and OS/VS COBOL programs, you must override the default AMODE setting to AMODE(24) when the load module contains a IBM COBOL program compiled with NORENT. (For programs compiled with RENT, no action is necessary. The linkage editor automatically assigns the correct AMODE setting.)

How you override the defaults: To override the defaults, specify AMODE or RMODE with one of the following:

- EXEC statement of your link-edit job step

  ```
  //LKED    EXEC    PGM=programname,
  //                PARM='AMODE=xx,RMODE=yy'
  ```

- The linkage editor mode control statements

  ```
  MODE AMODE(xx),RMODE(yy)
  ```

- One of the following TSO commands LINK or LOADGO

  LINK(*dsn-list*) AMODE(*xx*) RMODE(*yy*)

  LOADGO(*dsn-list*) AMODE(*xx*) RMODE(*yy*)

See your *Linkage Editor and Loader User's Guide* for allowable *xx* and *yy* and linkage editor mode control statements.

MVS uses a program's AMODE attribute to determine whether a program invoked using ATTACH, LINK, XCTL, or LOAD/BASSM is to receive control in 24-bit or 31-bit addressing mode. MVS uses the RMODE attribute to determine whether a program must be loaded into virtual storage below 16 megabytes, or can reside anywhere in virtual storage (above or below 16 megabytes).

Appendix K. Link-edit example

This appendix provides an example of JCL that shows how to replace the current library routines in a load module with the Language Environment library routines. The SCEESAMP data set contains 3 sample jobs (IGZWRLKA, IGZWRLKB, and IGZRLKC) to assist in relink-editing OS/VS COBOL or VS COBOL II load modules.

```
//*******************************************************************
//*                                                                 *
//*  RELINK A LOAD MODULE THAT HAS BOTH OS/VS COBOL PROGRAMS        *
//*  AND VS COBOL II PROGRAMS WITH Language Environment.             *
//*                                                                 *
//*******************************************************************
//*
//LINK   EXEC  PGM=IEWL,PARM='LIST,MAP,XREF'
//SYSPRINT DD  SYSOUT=*
//*******************************************************************
//*   CHANGE 'ZZZZZZ.SCEELKED' IN THE FOLLOWING DD STATEMENT TO     *
//*   THE Language Environment SCEELKED DATA SET NAME.              *
//*                                                                 *
//*   CHANGE 'XXXXXX' IN THE FOLLOWING DD STATEMENT TO              *
//*   THE DATA SET NAME WHICH CONTAINS THE LOAD MODULE.             *
//*                                                                 *
//*   CHANGE 'YYYYYY' IN THE FOLLOWING DD STATEMENT TO              *
//*   THE DATA SET NAME WHICH THE RELINK-EDITED LOAD                *
//*   MODULE SHOULD BE SAVED INTO.                                  *
//*                                                                 *
//*******************************************************************
//SYSLIB   DD  DSN=ZZZZZZ.SCEELKED,DISP=SHR
//LOADLIB  DD  DSN=XXXXXX,DISP=SHR
//SYSLMOD  DD  DSN=YYYYYY,DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(2,2)),DISP=NEW
//*******************************************************************
//*   CHANGE 'UUUUUU' IN THE FOLLOWING INCLUDE STATEMENT            *
//*   TO THE LOAD MODULE NAME.                                      *
//*                                                                 *
//*   CHANGE 'VVVVVV' IN THE FOLLOWING NAME STATEMENT TO            *
//*   THE RELINK-EDITED LOAD MODULE NAME.                           *
//*                                                                 *
//*   CHANGE 'EEEEEE' IN THE FOLLOWING ENTRY STATEMENT TO           *
//*   THE RELINK-EDITED LOAD MODULE ENTRY POINT NAME OR             *
//*   OMIT THE ENTRY STATEMENT IF IT IS NOT REQUIRED.               *
//*                                                                 *
//*******************************************************************
//SYSLIN   DD  *
  REPLACE ILBOxxxx  1
    .
    .
    .
  REPLACE IGZxxxx  2
    .
    .
    .
  INCLUDE LOADLIB(UUUUUU)
  ENTRY   EEEEEE
  NAME    VVVVVV(R)
/*
```

Where 1 and 2 represent the following:

1

```
REPLACE ILBOABN
REPLACE ILBOACP
REPLACE ILBOACS
REPLACE ILBOANE
REPLACE ILBOANF
REPLACE ILBOATB
REPLACE ILBOBEG
REPLACE ILBOBID
REPLACE ILBOBIE
REPLACE ILBOBII
REPLACE ILBOBUG
REPLACE ILBOCHN
REPLACE ILBOCJS
REPLACE ILBOCKP
REPLACE ILBOCLS
REPLACE ILBOCMM
REPLACE ILBOCOM0
REPLACE ILBOCT1
REPLACE ILBOCVB
REPLACE ILBODBE
REPLACE ILBODBG
REPLACE ILBODCI
REPLACE ILBODSP
REPLACE ILBODSS
REPLACE ILBODTE
REPLACE ILBOD01
REPLACE ILBOD10
REPLACE ILBOD11
REPLACE ILBOD12
REPLACE ILBOD13
REPLACE ILBOD14
REPLACE ILBOD20
REPLACE ILBOD21
REPLACE ILBOD22
REPLACE ILBOD23
REPLACE ILBOD24
REPLACE ILBOD25
REPLACE ILBOD26
REPLACE ILBOEFL
REPLACE ILBOERR
REPLACE ILBOETB
REPLACE ILBOEXT
REPLACE ILBOFLW
REPLACE ILBOFPW
REPLACE ILBOGDO
REPLACE ILBOGPW
REPLACE ILBOIDB
REPLACE ILBOIDR
REPLACE ILBOIDT
REPLACE ILBOIFB
REPLACE ILBOIFD
REPLACE ILBOINS
REPLACE ILBOINT
REPLACE ILBOITB
REPLACE ILBOIVL
REPLACE ILBOLBL
REPLACE ILBOMRG
REPLACE ILBOMSG
REPLACE ILBOMSC
REPLACE ILBONBL
REPLACE ILBONED
REPLACE ILBONTR
REPLACE ILBOOCR
REPLACE ILBOPRM
REPLACE ILBOPTV
REPLACE ILBOQIO
REPLACE ILBOQSS
REPLACE ILBOQSU
REPLACE ILBOREC
REPLACE ILBORNT
REPLACE ILBOSAM
REPLACE ILBOSCD
REPLACE ILBOSCH
REPLACE ILBOSDB
REPLACE ILBOSGM
REPLACE ILBOSMG
REPLACE ILBOSMV
REPLACE ILBOSND
REPLACE ILBOSNT
REPLACE ILBOSPI
REPLACE ILBOSPN
REPLACE ILBOSPA
REPLACE ILBOSQA
REPLACE ILBOSRT
REPLACE ILBOSRV
REPLACE ILBOSSN
REPLACE ILBOSTG
REPLACE ILBOSTI
REPLACE ILBOSTN
REPLACE ILBOSTR
REPLACE ILBOSTT
REPLACE ILBOSYN
REPLACE ILBOTC0
REPLACE ILBOTC2
REPLACE ILBOTC3
REPLACE ILBOTEF
REPLACE ILBOTRN
REPLACE ILBOUST
REPLACE ILBOUTB
REPLACE ILBOVCO
REPLACE ILBOVIO
REPLACE ILBOVMO
REPLACE ILBOVOC
REPLACE ILBOVTR
REPLACE ILBOWAT
REPLACE ILBOWTB
REPLACE ILBOXDI
REPLACE ILBOXMU
REPLACE ILBOXPR
```

2

```
REPLACE IGZCA2D
REPLACE IGZCACP
REPLACE IGZCACS
REPLACE IGZCANE
REPLACE IGZCANF
REPLACE IGZCBID
REPLACE IGZCBUG
REPLACE IGZCCCO
REPLACE IGZCCLS
REPLACE IGZCCTL
REPLACE IGZCCVB
REPLACE IGZCD2A
REPLACE IGZCDIF
REPLACE IGZCDSP
REPLACE IGZCFDP
REPLACE IGZCFDW
REPLACE IGZCFPW
REPLACE IGZCGDR
REPLACE IGZCIDB
REPLACE IGZCINS
REPLACE IGZCIN1
REPLACE IGZCIN2
REPLACE IGZCIPS
REPLACE IGZCIVL
REPLACE IGZCKCL
REPLACE IGZCLNK
REPLACE IGZCMSF
REPLACE IGZCMST
REPLACE IGZCONV
REPLACE IGZCONVX
REPLACE IGZCSCH
REPLACE IGZCSMV
REPLACE IGZCSPA
REPLACE IGZCSPC
REPLACE IGZCSSN
REPLACE IGZCSTG
REPLACE IGZCTCO
REPLACE IGZCULE
REPLACE IGZCUST
REPLACE IGZCVIN
REPLACE IGZCVMO
REPLACE IGZCXDI
REPLACE IGZCXFR
REPLACE IGZCXMU
REPLACE IGZCXPR
REPLACE IGZEABX
REPLACE IGZEABN
REPLACE IGZEBRG
REPLACE IGZEBST
REPLACE IGZECKP
REPLACE IGZECMS
REPLACE IGZEDBR
REPLACE IGZEDBW
REPLACE IGZEDTE
REPLACE IGZEINP
REPLACE IGZEMSG
REPLACE IGZENRI
REPLACE IGZENRT
REPLACE IGZEOPD
REPLACE IGZEOPT
REPLACE IGZEOUT
REPLACE IGZEPRM
REPLACE IGZEPTV
REPLACE IGZEQBL
REPLACE IGZEQOC
REPLACE IGZERRE
REPLACE IGZESAT
REPLACE IGZESMG
REPLACE IGZESNP
REPLACE IGZESPM
REPLACE IGZESTA
REPLACE IGZETRM
REPLACE IGZETUN
REPLACE IGZEVAM
REPLACE IGZEVEX
REPLACE IGZEVIO
REPLACE IGZEVOC
REPLACE IGZEVOP
REPLACE IGZEVSV
REPLACE IGZTCAM2
REPLACE IGZTCAM4
REPLACE IGZTCM21
REPLACE IGZTCM41
REPLACE IGZTCM42
```

Appendix L. Differences between CMPR2 and NOCMPR2

IBM COBOL with the NOCMPR2 compiler option provides COBOL 85 Standard support, while VS COBOL II Release 2 provides COBOL 74 Standard support. The implementation of the COBOL 85 Standard caused some language elements to behave in a manner that differs from the implementation of the COBOL 74 Standard.

Beginning with VS COBOL II Release 3.0, you could choose COBOL 85 Standard behavior (without the Intrinsic Function module) by using NOCMPR2, or the COBOL 74 Standard behavior by using the CMPR2 compiler option. The CMPR2 option provides COBOL 74 Standard-conforming behavior as implemented by VS COBOL II Release 2, as well as **nonstandard** Release 2 extensions now implemented in the COBOL 85 Standard. The NOCMPR2 option provides COBOL 85 Standard-conforming behavior and IBM extensions. This same mechanism is provided by IBM COBOL as an aid in upgrading your existing CMPR2 programs to IBM COBOL NOCMPR2.

When referring to VS COBOL II and IBM COBOL, the following terms have been defined:

CMPR2

We use CMPR2 to refer to the language and behavior of programs compiled and run with:

- VS COBOL II Release 2
- VS COBOL II Release 4 with the CMPR2 compiler option
- IBM COBOL with the CMPR2 compiler option.

NOCMPR2

We use NOCMPR2 to refer to the language and behavior of programs compiled and run with:

- VS COBOL II Release 4 with the NOCMPR2 compiler option
- IBM COBOL with the NOCMPR2 compiler option.

Standard changes: The language differences described in this chapter are a result of changes in the COBOL Standard. Thus, this information also applies to COBOL/370 programs.

CMPR2 problem: This book is designed to help you move from CMPR2 to NOCMPR2. If you have programs that compile with NOCMPR2, do not use the CMPR2 option. Doing so will cause problems that this book does not provide help for.

The language elements listed below are affected by the IBM COBOL CMPR2/NOCMPR2 compiler option. The differences are explained in the sections that follow.

Figure 58 (Page 1 of 2). Language elements different between CMPR2 and NOCMPR2

Language element	Page
ALPHABET clause of the SPECIAL-NAMES paragraph	248
ALPHABETIC class	249
CALL...ON OVERFLOW	249

Figure 58 (Page 2 of 2). Language elements different between CMPR2 and NOCMPR2

Language element	Page
Comparisons between scaled integers and nonnumerics	251
COPY...REPLACING statements using non-COBOL characters	252
COPY statement using national extension characters	254
File status codes	255
Implicit EXIT PROGRAM	256
PERFORM return mechanism	257
PERFORM...VARYING...AFTER	260
PICTURE clause with "A"s and "B"s	262
PROGRAM COLLATING SEQUENCE	264
READ INTO and RETURN INTO	266
RECORD CONTAINS n CHARACTERS	267
Reserved words	268
SET...TO TRUE	269
SIZE ERROR on MULTIPLY and DIVIDE	271
UNSTRING operand evaluation	272
UPSI switches	279
Variable-length group moves	280

ALPHABET clause of the SPECIAL-NAMES paragraph

CMPR2

The ALPHABET clause does not include the keyword ALPHABET. In fact, ALPHABET is not a reserved word.

For example:

```
SPECIAL-NAMES.
   ALPHA-NAME IS STANDARD-1.
```

NOCMPR2

The ALPHABET clause requires the use of the keyword ALPHABET. ALPHABET is now a reserved keyword.

For example:

```
SPECIAL-NAMES.
   ALPHABET ALPHA-NAME IS STANDARD-1.
```

Messages

Compiling the program with the CMPR2 and FLAGMIG compiler options will generate the following message for each ALPHABET clause of the SPECIAL-NAMES paragraph:

IGYDS1190-W **MIGR** Alphabet-name must be preceded by the keyword "ALPHABET" under the "NOCMPR2" compiler option.

Corrective action
Add the keyword ALPHABET to the ALPHABET clause.

ALPHABETIC class

CMPR2
The ALPHABETIC class of characters defined by the ALPHABETIC class test consists of the 26 uppercase letters and the space. The 26 lowercase letters are not considered alphabetic.

For example:

```
MOVE "AbC dE" TO PIC-X6.
IF PIC-X6 IS NOT ALPHABETIC THEN DISPLAY "CMPR2".
```

NOCMPR2
The ALPHABETIC class of characters defined by the ALPHABETIC class test consists of the 26 uppercase letters, the 26 lowercase letters, and the space.

For example:

```
MOVE "AbC dE" TO PIC-X6.
IF PIC-X6 IS ALPHABETIC THEN DISPLAY "NOCMPR2".
```

Messages
Compiling the program with the CMPR2 and FLAGMIG compiler options will generate the following message for each ALPHABETIC class test:

IGYPS2221-W **MIGR** The alphabetic class has been expanded to include lowercase letters under the "NOCMPR2" compiler option.

Corrective action
Use the ALPHABETIC-UPPER class test under NOCMPR2 to get the same function as the ALPHABETIC class test under CMPR2. The ALPHABETIC-UPPER class under NOCMPR2 consists of the 26 uppercase letters and the space.

CALL...ON OVERFLOW

CMPR2
Under CMPR2, the ON OVERFLOW condition exists if the available portion of object time memory cannot accommodate the program specified in the CALL statement. VS COBOL II Release 2 interpreted that definition to cover only the condition "not enough storage available to load the program."

Note: Only errors that occur on the actual LOAD of the called program raise the ON OVERFLOW condition. Errors occurring after the program has been loaded and has started execution do not raise the condition.

NOCMPR2
Under NOCMPR2, the ON OVERFLOW condition exists if the program specified by the CALL statement cannot be made available for execution at that time.

NOCMPR2 implements the COBOL 85 Standard rules and defines the ON OVERFLOW condition to handle any "recoverable" condition that may prevent the called program from being made available.

Note: Only errors that occur on the actual LOAD of the called program raise the ON OVERFLOW condition. Errors occurring after the program has been loaded and started execution do not raise the condition.

Messages

Compiling the program with the CMPR2 and FLAGMIG options will cause the compiler to issue messages for all CALL statements that specify the ON OVERFLOW phrase. The following message will be issued:

IGYPS2012-W **MIGR** The "ON OVERFLOW" phrase of the "CALL" statement will execute under more conditions under the "NOCMPR2" compiler option.

The following program fragment illustrates one situation that will be affected by this change:

```
PERFORM UNTIL ALL-ACCOUNTS-SETTLED
  ⋮
  CALL "SUBPROGA" USING CURRENT-ACCOUNT
    ON OVERFLOW
       CANCEL "SUBPROGB"
       CALL "SUBPROGA" USING CURRENT-ACCOUNT
       END-CALL
  END-CALL
  ⋮
  CALL "SUBPROGB" USING CURRENT-ACCOUNT
    ON OVERFLOW
       CANCEL "SUBPROGA"
       CALL "SUBPROGB" USING CURRENT-ACCOUNT
       END-CALL
  END-CALL
  ⋮
END-PERFORM
```

The assumption is that for some executions of this program, SUBPROGA and SUBPROGB may not fit into available storage at the same time.

The ON OVERFLOW phrase is used to react to this situation, and to release the storage occupied by the other subprogram.

Running under a program that exhibits CMPR2 behavior, the ON OVERFLOW condition will be raised only for the "out of storage" errors, and the above approach is reasonable.

Running under NOCMPR2, the ON OVERFLOW condition may be raised for errors other than the "out of storage" errors, and therefore, the second call (inside the ON OVERFLOW phrase) may fail as well.

Corrective action

No correction that is generally applicable exists for programs using this or similar techniques. If the ON OVERFLOW condition is indeed raised because of the "out of storage" error, the program will exhibit the same behavior as before; if the condition is raised for some other error, the recovery code (provided in the ON OVERFLOW phrase) may not correct the error, and the subsequent CALL will fail as well.

In general, it is not possible for an IBM COBOL program to determine the actual cause of the error that raised the ON OVERFLOW condition.

Comparisons between scaled integers and nonnumerics

In comparisons between nonnumeric items and numeric items (only integers are allowed), the value of the numeric item used in the comparison will differ if it is scaled.

CMPR2

Under CMPR2, the numeric or algebraic value of a scaled numeric item is used in comparison operations with nonnumeric items. In determining the algebraic value, all symbols P in the PICTURE character-string are included in the total number of digits, and zeros are used in their place.

NOCMPR2

Under NOCMPR2, the actual character representation or character value of the scaled numeric item is used in comparison operations with nonnumeric items. The character value for scaled numeric items does not include any digit positions specified with the symbol P. These digit positions are ignored and not counted in the size of the operand.

For example:

```
01  NUM    PIC 99PP  VALUE 2300.
01  ALPHA1 PIC XX    VALUE "23".
01  ALPHA2 PIC XXX   VALUE "23".
01  ALPHA3 PIC XXXX  VALUE "2300".

    IF NUM EQUAL ALPHA1 DISPLAY "ALPHA1".
    IF NUM EQUAL ALPHA2 DISPLAY "ALPHA2".
    IF NUM EQUAL ALPHA3 DISPLAY "ALPHA3".
```

	CMPR2	NOCMPR2
Results displayed	ALPHA3	ALPHA1 ALPHA2

In this example, under NOCMPR2, the character value of NUM has only two digit positions. When it is compared to a nonnumeric item of unequal length as in ALPHA2, the shorter operand (NUM) is padded with enough blanks to equal the length of the other operand.

Messages

Compiling a program with the CMPR2 and FLAGMIG options will cause the compiler to issue the following message for all comparisons between scaled integers and nonnumeric items.

IGYPG3138-W **MIGR** The comparison between the scaled integer item " " and the nonnumeric item " " will be performed differently under the "NOCMPR2" compiler option.

Corrective action

To preserve CMPR2 behavior, you can define the scaled integer within a structure. FILLER serves as the placeholders for the integer scaling positions and must be initialized to zero. There must be as many alphanumeric positions defined in FILLER as there are scaling positions in NUM. Wherever NUM is used in a comparison with a nonnumeric item, CHARVAL should be substituted instead.

```
01  CHARVAL.
  05  NUM     PIC 99PP  VALUE 2300.
  05  FILLER  PIC XX    VALUE "00".

      IF CHARVAL EQUAL ALPHA1 DISPLAY "ALPHA1".
      IF CHARVAL EQUAL ALPHA2 DISPLAY "ALPHA2".
      IF CHARVAL EQUAL ALPHA3 DISPLAY "ALPHA3".
```

COPY...REPLACING statements using non-COBOL characters

This section describes three classes of "non-COBOL" characters occurring in library text or COPY...REPLACING statements that are treated differently under NOCMPR2 than under CMPR2.

"Non-COBOL" characters are the EBCDIC characters outside the legal set of COBOL characters, excluding nonnumeric literals. Nonnumeric literals can contain any character within the character set of the computer.

CMPR2

Under CMPR2, library text and COPY...REPLACING statements can contain operands consisting of "non-COBOL" characters.

NOCMPR2

The COBOL 85 Standard disallows all "non-COBOL" characters and adds "lowercase" and the colon to the character set.

Corrective action

"Lowercase" alphabetic characters

"Lowercase" alphabetic characters, which were "non-COBOL" in VS COBOL II Release 2, are now in the set of legal COBOL characters in IBM COBOL. In VS COBOL II Release 2, COPY allowed replacement of lowercase characters:

```
COPY  A  REPLACING == abc ==  BY  == XYZ ==.
```

The previous example would locate all instances of "abc" and replace it with "XYZ". In contrast, IBM COBOL will treat lowercase and uppercase characters as equivalent in data-names and replace all instances of "abc" as well as "ABC" with "XYZ". If member A contains:

```
1  abc PIC X.
1  ABC PIC XX.
```

then the results are as follows:

```
   VS COBOL II R2              IBM COBOL NOCMPR2

After COPYing & REPLACING     After COPYing & REPLACING
 1  XYZ PIC X.                    1  XYZ  PIC X.
 1  ABC PIC XX.                   1  XYZ  PIC XX.
```

To obtain the same results when running VS COBOL II Release 2 programs under IBM COBOL, you can verify that all your REPLACING arguments are unique (even after folding to uppercase) or you can use the CMPR2 option.

Message

The difference in behavior is flagged by the FLAGMIG compiler option.

IGYLI0161-W **MIGR** Lowercase character " " found in column " " will be treated the same as its uppercase equivalent under the "NOCMPR2" compiler option. Results may be different.

The colon (:) character

In VS COBOL II Release 2, the colon was a non-COBOL character that COPY...REPLACING allowed as part of its operands. This character is a legal COBOL separator under IBM COBOL NOCMPR2.

```
COPY  A  REPLACING  == A ==    BY  == X ==
                    == B ==    BY  == Y ==
                    == A:B ==  BY  == Z ==.
```

If member A contains:

```
MOVE  A:B  TO  ID2.
```

Then the following are the differences between VS COBOL II Release 2 and IBM COBOL NOCMPR2 after the COPYing and REPLACING.

```
 VS COBOL II R2              IBM COBOL NOCMPR2

MOVE  Z  TO  ID2.             MOVE  X:Y  TO  ID2.
```

Because ":" is a separator under IBM COBOL NOCMPR2, "A:B" is broken up into three separate tokens: "A" ":" and "B." The replacements for A and B are made first.

To make the previous piece of code behave in the same manner as in VS COBOL II Release 2, change the REPLACING clauses to:

```
COPY  A  REPLACING  == A:B ==  BY  == Z ==
                    == A ==    BY  == X ==
                    == B ==    BY  == Y ==.
```

Message

This difference in behavior between the two releases is also flagged by FLAGMIG.

IGYLI0160-W **MIGR** The colon will be treated as a separator under the "NOCMPR2" compiler option. Results may be different.

"Invalid" characters

"Invalid" characters do not fall into the legal COBOL character set. Consider this example:

```
COPY  A  REPLACING  == % ==  BY  == 1 ==.
```

where member A contains:

```
%  XDATA  PIC  X.
```

Here, the "non-COBOL" character is the "%" character.

Under both CMPR2 and NOCMPR2, the above member will be copied with the replacement executed. Under NOCMPR2, the compiler will issue an E-level diagnostic.

IGYLI0163-E Non-COBOL character "%" was found in column 8. The character was accepted.

In both cases, after processing all COPY statements, a legal COBOL program should result.

Message

This difference in behavior between the two releases is also flagged by FLAGMIG.

IGYLI0162-W **MIGR** Non-COBOL character "%" found in column 8 will be diagnosed under the "NOCMPR2" compiler option. Results may be different.

To protect yourself against new problems in later releases of IBM COBOL, you should remove all non-COBOL characters from your source programs and COPY libraries, and replace them with COBOL characters. Future releases may assign meaning to one of these characters (as with the colon) and results might be different.

COPY statement using national extension characters

CMPR2

National extension characters @, #, and $ are allowed in the text-name and library-name of the COPY statement. For example, `COPY X$3.`: will be copied.

NOCMPR2

The compiler will issue an E-level diagnostic.

IGYLI0025-E

Name "X$3" was invalid. It was processed as "X03".

IBM COBOL allows national extension characters @, #, and $ in the text-name and library-name, if they are in the form of a nonnumeric literal. For example, to copy X$3 in IBM COBOL, code: `COPY "X$3".`.

Message

The difference in behavior is also flagged by FLAGMIG.

IGYLI0115-W

MIGR The name "X$3" did not follow the rules for formation of a program-name. It will be diagnosed under the "NOCMPR2" compiler option.

File status codes

CMPR2

Under CMPR2, VS COBOL II Release 2 file status codes are returned.

NOCMPR2

Under NOCMPR2, the file status codes are enhanced. New and changed file status codes are returned and more detail is provided about the status of I-O operations. In addition, problems are detected earlier in some cases and there are updated definitions and file status conditions for "missing" files.

Message

A program that contains a file status data-name will receive the following message when compiled with the CMPR2 and FLAGMIG compiler options:

IGYGR1188-W **MIGR** The file status values are different under the "NOCMPR2" compiler option.

Corrective action

Although there is no one-to-one mapping of the VS COBOL II Release 2 status codes to those in IBM COBOL, the following tables show, in general, the relationships between CMPR2 and NOCMPR2 file status codes. For a comprehensive definition of the NOCMPR2 file status codes, see the *COBOL Language Reference*.

Figure 59 (Page 1 of 2). QSAM and VSAM file status codes with CMPR2 and NOCMPR2

VSAM file status codes		QSAM file status codes	
CMPR2	**NOCMPR2**	**CMPR2**	**NOCMPR2**
00	00 04 05 14 24 35 39 44	00	00 04 05 07 39 44
02	02	10	10
10	10	30	30 39
21	21	34	34
22	22	90	35 37 90
23	23	92	38 41 42 43 46 47 48 49 92
24	24		
30	30 39		

Figure 59 (Page 2 of 2). QSAM and VSAM file status codes with CMPR2 and NOCMPR2

VSAM file status codes		QSAM file status codes	
CMPR2	**NOCMPR2**	**CMPR2**	**NOCMPR2**
90	37 90		
91	91		
92	38 41 42 43 44 47 48 49 92		
93	93		
94	46		
95	39 95		
96	96		
97	97		

Implicit EXIT PROGRAM

To end a program, you must use an EXIT PROGRAM, STOP RUN, or GOBACK statement. You can use an EXIT PROGRAM for a called subprogram; you can use a STOP RUN for a main program. GOBACK, an IBM extension, can be used for either type of program.

CMPR2

Under CMPR2, if a program does not contain any of the above statements, a compiler warning diagnostic message will be issued to suggest that the program should be analyzed to verify that it could exit.

Suppose that this is the last line in the program:

```
IF TALLY = 0 THEN STOP RUN.
```

In this case, the compiler diagnostic message would not be issued, and the following run-time message would be issued only if the IF condition tested false:

IGZ0037S The flow of control in program "program-name" proceeded beyond the last line of the program.

NOCMPR2

Under NOCMPR2, all programs are assumed to end with an EXIT PROGRAM statement. For a called subprogram, then, control can no longer flow beyond the last line of the program, but instead, the program will return to the calling program. In the preceding example, where the program ended with the statement:

```
IF TALLY = 0 THEN STOP RUN.
```

a false test will cause control to be returned to the caller. With CMPR2 behavior, the result is an ABEND.

For a main program, the EXIT PROGRAM statement has no effect. Therefore, the implicit EXIT PROGRAM that is generated by the compiler will have no effect on

the execution of the program; a main program that executes beyond the last line of the program will still ABEND.

Messages

A program that does not contain a STOP RUN, GOBACK, or EXIT PROGRAM statement will receive the following diagnostic message:

IGYPS2091-W No "STOP RUN", "GOBACK" or "EXIT PROGRAM" was found in the program. Check program logic to verify that the program will exit.

Also, if the CMPR2 and FLAGMIG compiler options are used, the following message will appear:

IGYPS2223-W **MIGR** An implicit "EXIT PROGRAM" will be executed at the end of this program under the "NOCMPR2" compiler option.

If a program does contain a STOP RUN, GOBACK, or EXIT PROGRAM statement, and the NOOPTIMIZE compiler option is in effect, then use of the FLAGMIG compiler option will result in the following message:

IGYPS2224-W **MIGR** An implicit "EXIT PROGRAM" may be executed at the end of this program under the "NOCMPR2" compiler option. Recompile with the "OPTIMIZE" and "FLAGMIG" compiler options. If no "MIGR" message about an implicit "EXIT PROGRAM" is issued then no implicit "EXIT PROGRAM" will be executed.

Upon re-compilation with the OPTIMIZE compiler option, the absence of any such messages indicates that the program will not have an implicit EXIT PROGRAM generated for it, while the presence of the following message indicates otherwise:

IGYOP3210-W **MIGR** An implicit "EXIT PROGRAM" will be executed at the end of this program under the "NOCMPR2" compiler option.

Corrective action

To preserve CMPR2 behavior, a program can be modified to contain a new section and section-name as the very last section in the program. That new section can then contain error-handling code, such as a call to "ILBOABN0".

Any program receiving a message indicating that an EXIT PROGRAM will be implicitly generated should be examined to ensure that it will exit properly.

PERFORM return mechanism

When a paragraph or a range of paragraphs is executed with a PERFORM statement ("out-of-line PERFORM"), a mechanism at the end of the range of paragraphs causes control to be returned to the point just after the PERFORM statement.

Consider the following example:

```
    PERFORM A
    STOP RUN.
A.  DISPLAY "Hi".
B.  DISPLAY "there".
```

After displaying the message "Hi," compiler-generated code will cause the flow of control to return to the STOP RUN statement after performing paragraph A. Without this, control would fall through into paragraph B.

This code mechanism is reset to an initial state the first time a program is called or when a program is cancelled. Under NOCMPR2, it is also reset every time a program is called. Under CMPR2, the mechanism retains its last used state when a program is called twice in succession without having been cancelled. This can be important when the program issues an EXIT PROGRAM or GOBACK statement before all of the PERFORM statements have completed their execution.

Now consider this example:

```
    IF FIRST-TIME-CALLED THEN
      PERFORM A
      MOVE ZERO TO N
    ELSE
      SUBTRACT 1 FROM N
      GO TO A.
    GOBACK.
A.  IF N > 1 THEN
      GOBACK.
B.  DISPLAY "Processing continues...".
```

The program is passed a switch, FIRST-TIME-CALLED, which tells the program whether or not the program has been called without having been cancelled. It is also passed a variable, N.

CMPR2

When the program is called for the first time, the PERFORM statement will be executed. If the "N > 1" test succeeds, the program will return to the calling program.

However, this means that the PERFORM statement has not reached normal completion because paragraph A never returned to the point from which it was performed. The compiler-generated mechanism at the end of paragraph A is still "set" to return back to the PERFORM statement.

Thus, on the second call to the program, the ELSE path will be taken, 1 will be subtracted from N, and control will be transferred by the GO TO statement to paragraph A. However, if the test "N > 1" fails, the PERFORM mechanism is still set. So, when the end of paragraph A is reached, instead of falling through into paragraph B, control is "returned" to the MOVE statement after the PERFORM statement.

These results may not be intended. The problem may occur whenever all of the following occur:

1. The program returns to the calling program with an EXIT PROGRAM or GOBACK statement.
2. A PERFORM statement performs a paragraph or a range of paragraphs, and those paragraphs may also be reached by a GO TO statement or by falling through into the paragraph.
3. All such PERFORM statements have not had a chance to return prior to the execution of the EXIT PROGRAM or GOBACK statement.

NOCMPR2

Under NOCMPR2, when the program is called for the first time, the PERFORM statement will be executed and control will flow to paragraph A. Then, depending on the result of the test "N > 1," the program will either immediately return to the calling program, or it will return back to the PERFORM, move zero to N, and then return to the calling program.

On subsequent calls to the program, the ELSE path will be taken, 1 will be subtracted from N, and then control will be transferred by the GO TO statement to paragraph A. Then, depending on the result of the test "N > 1," the program will either immediately return to the calling program or fall through into paragraph B, display a message, and continue.

Regardless of the paths taken, the mechanism that controls the PERFORM statement will be reset each time the program is called and no irregular control flow will take place.

Messages

A program that contains an out-of-line PERFORM, and either an EXIT PROGRAM or GOBACK statement, will receive the following messages when compiled with the CMPR2, FLAGMIG and NOOPTIMIZE compiler options:

IGYPA3205-W **MIGR** "EXIT PROGRAM" or "GOBACK" statements assume that ends of "PERFORM" ranges were reached under the "NOCMPR2" compiler option. This program may have different execution results after migration if used as a subprogram.

IGYPA3206-W **MIGR** For more information about ends of "PERFORM" ranges, recompile with the "OPTIMIZE" and "FLAGMIG" compiler options. If no messages about ends of "PERFORM" ranges are issued, then this program will not have a migration problem with ends of "PERFORM" ranges.

Upon re-compilation with the OPTIMIZE compiler option, the absence of any such messages indicates that the program will not have any problem with an EXIT PROGRAM or GOBACK statement being executed within the range of an out-of-line PERFORM statement, while the presence of the following messages indicates otherwise:

IGYOP3205-W **MIGR** "EXIT PROGRAM" or "GOBACK" statements assume that ends of "PERFORM" ranges were reached under the "NOCMPR2" compiler option. This program may have different execution results after migration if used as a subprogram.

IGYOP3092-W An "EXIT PROGRAM" or a "GOBACK" statement was encountered in the range of the "PERFORM" statement at "PERFORM (LINE xx.xx)". Re-entry of the program may cause unexpected control flow.

Corrective action

The CMPR2 behavior of affected programs cannot be preserved without extensive and complex recoding. Such programs should be rewritten to avoid this dependency on the CMPR2 behavior.

PERFORM...VARYING...AFTER

Identifiers are set and incremented differently, for example:

```
PERFORM PARA3 VARYING id-2 FROM id-3 BY id-4
              UNTIL condition-1
        AFTER id-5 FROM id-6 BY id-7
              UNTIL condition-2.
```

CMPR2

Within the VARYING...AFTER phrase of the PERFORM statement under CMPR2, id-5 is set before id-2 is augmented.

When varying two variables under CMPR2, at the intermediate stage when the inner condition is true, the inner variable (id-5) was set to its current FROM value (id-6) before the outer variable (id-2) was augmented with its current BY value (id-4).

NOCMPR2

However, under NOCMPR2, id-2 is augmented before id-5 is set. This change creates an incompatibility when id-6 is dependent on id-2.

Consider the following example:

```
PERFORM PARA3 VARYING X FROM 1 BY 1 UNTIL X IS GREATER THAN 3
          AFTER Y FROM X BY 1 UNTIL Y IS GREATER THAN 3.
```

In this example, id-6 (X) is dependent on id-2 (X) because they are identical.

Under CMPR2, PARA3 will be executed eight times with the following values:

```
X:    1   1   1   2   2   2   3   3
Y:    1   2   3   1   2   3   2   3
```

Under NOCMPR2, PARA3 will be executed six times with the following values:

```
X:    1   1   1   2   2   3
Y:    1   2   3   2   3   3
```

A dependency between identifiers occurs if the first identifier is identical to, subscripted with, a partial or full redefinition of, or variably located depending on the second identifier.

Message

First, recompile all programs under IBM COBOL with the CMPR2 and FLAGMIG compiler options. This will flag any PERFORM...VARYING statements that have dependencies between the following variables:

- id-6 is (potentially) dependent on id-2
- id-9 is (potentially) dependent on id-5
- id-4 is (potentially) dependent on id-5
- id-7 is (potentially) dependent on id-8

Note: Only PERFORM...VARYING with the AFTER phrase is affected.

For example, compiling the program under IBM COBOL with the CMPR2 and FLAGMIG compiler options will cause the compiler to issue the following message when id-6 is dependent on id-2:

IGYPA3209-W **MIGR** "PERFORM VARYING" operand "ID-6 (NUMERIC INTEGER)" was dependent on "ID-2 (NUMERIC INTEGER)". Under the "NOCMPR2" compiler option, the rules for augmenting/setting "PERFORM VARYING" operands have changed, and this statement may have incompatible results.

Corrective action

If a PERFORM...VARYING statement is flagged by FLAGMIG, that statement will have to be converted. A possible way of converting a PERFORM...VARYING statement that has **all four** dependencies is as follows:

```
PERFORM xx
   VARYING id-2 FROM id-3 BY id-4 UNTIL cond-1
     AFTER  id-5 FROM id-6 BY id-7 UNTIL cond-2
     AFTER id-8 FROM id-9 BY id-10 UNTIL cond-3.
```

is converted into:

```
MOVE id-3 TO id-2.
MOVE id-6 TO id-5
MOVE id-9 TO id-8

PERFORM UNTIL cond-1
  PERFORM UNTIL cond-2
    PERFORM UNTIL cond-3
      PERFORM xx
      ADD id-10 TO id-8
    END-PERFORM
    MOVE id-9 TO id-8
    ADD id-7 TO id-5
  END-PERFORM
  MOVE id-6 TO id-5
  ADD id-4 TO id-2
END-PERFORM
```

This example assumes that all id-x are identifiers. If any are index-names, then SET statements must be used in place of MOVE statements.

The above example is a worst-case conversion. It could be refined by changing only the parts of the statement that use those identifiers for which a dependency (potentially) exists. For example, if only id-6 is dependent on id-2 and no other dependency exists, the above conversion can be reduced to:

```
MOVE id-3 TO id-2.
MOVE id-6 TO id-5.

PERFORM UNTIL cond-1
  PERFORM UNTIL cond-2
    PERFORM VARYING  id-8 FROM id-9 BY id-10 UNTIL cond-3
         PERFORM XX
    END-PERFORM
    ADD id-7 TO id-5
  END-PERFORM
  MOVE id-6 TO id-5
  ADD id-4 TO id-2
END-PERFORM
```

PICTURE clause with "A"s and "B"s

CMPR2

Under CMPR2, a data item with the symbol B in its PICTURE clause is an alphabetic data item.

NOCMPR2

Under NOCMPR2, a data item with the symbol B in its PICTURE clause is an alphanumeric-edited item.

Most functions do not pose a problem with this change. However, there are a few subtleties that you should watch for when upgrading from VS COBOL II Release 2 to IBM COBOL NOCMPR2, relating to the INITIALIZE, STRING, CALL and CANCEL verbs.

Message

If a program is compiled with the CMPR2 and FLAGMIG options, a message will appear for any alphabetic items that had been defined with the symbol B.

IGYDS1105-W **MIGR** A "PICTURE" clause was found consisting of symbols "A" and "B". This alphabetic item will be treated as an alphanumeric-edited item under the "NOCMPR2" compiler option.

Corrective action

INITIALIZE verb

Consider the following example:

```
01  ALPHA  PIC AABAABAA.

INITIALIZE ALPHA REPLACING ALPHABETIC DATA BY ALL "3".
```

A statement like this coded under CMPR2 is valid and initialization will take place. However, this statement gives the following warning message under NOCMPR2, and no initialization will take effect:

IGYPS2047-W "INITIALIZE" statement receiver "ALPHA" was incompatible with the data category(s) of the "REPLACING" operand(s). "ALPHA" was not initialized.

This can also happen when a group of items are being initialized. Under NOCMPR2, ALPHA above would be classified as alphanumeric-

edited. If ALPHA was defined in a group that was to be initialized, a message like the one above would appear only if there were no alphabetic items to be initialized. Thus, in the following example, ALPHA is never initialized, but no message alerts you to that fact.

```
01  GROUP1.
    05 ALPHA  PIC AABAA.
    05 BETA   PIC AAA.

INITIALIZE GROUP1 REPLACING ALPHABETIC DATA BY ALL "5".
```

To initialize any of these reclassified data items in the same manner they had been previously, change the original statement for the first example above to the following:

```
INITIALIZE ALPHA REPLACING
           ALPHANUMERIC-EDITED DATA BY ALL "3".
```

In the second example in which a group of possibly mixed types are involved, INITIALIZE should be supplemented with an additional phrase. For example:

```
INITIALIZE GROUP1 REPLACING
                  ALPHABETIC DATA BY ALL "5"
                  ALPHANUMERIC-EDITED DATA BY ALL "5".
```

Note: Adding this extra phrase could cause conflicts if you had already specified this phrase but used different replacing data **or** if you had other alphanumeric-edited items within the group that you did not want initialized.

STRING verb

With either CMPR2 or NOCMPR2, alphabetic items are allowed to be the STRING...INTO receiving field. However, edited items are not allowed. Therefore, if any VS COBOL II Release 2, programs have an alphabetic item defined with the symbol B in this position of the STRING verb, these programs will get a severe error message because this item is reclassified as alphanumeric-edited.

IGYPA3104-S "STRING INTO" identifier "ALPHA (ALPHANUMERIC-EDITED)" was an edited data item or was defined with the "JUSTIFIED" clause. The statement was discarded.

Because a STRING statement in VS COBOL II Release 2 would automatically overlay any positions represented with the symbol B, all that is really needed is a new alphabetic data-name redefined on the original INTO field. For example:

Statement under CMPR2:

```
01 ALPHA  PIC AABAABAA.
01 VARX   PIC A(3)  VALUE "XXX".
01 VARY   PIC A(3)  VALUE "YYY".

   STRING VARX VARY DELIMITED BY SIZE INTO ALPHA.
```

Statement under NOCMPR2:

```
01 ALPHA  PIC AABAABAA.
01 BETA   REDEFINES ALPHA   PIC A(8).
01 VARX   PIC A(3)  VALUE "XXX".
01 VARY   PIC A(3)  VALUE "YYY".

   STRING VARX VARY DELIMITED BY SIZE INTO BETA.
```

BETA is redefined on ALPHA and has a length equal to ALPHA, including all symbols of B. BETA is then used in the STRING statement. After STRING is executed, ALPHA will have the same value as it did in Release 2.

CALL and CANCEL verbs

An IBM extension allows the CALL and CANCEL statement identifier to be an alphabetic data item. However, alphanumeric-edited items are not allowed; therefore, any VS COBOL II Release 2 programs with alphabetic items defined with the symbol B will get a severe error message. For example, the following program would have worked in VS COBOL II Release 2, but will now get a severe error message:

```
01  CALLDN   PIC  AAAAABB.

      MOVE "PROG1" TO CALLDN.
      CALL CALLDN.
      CANCEL CALLDN.
```

IGYPA3063-S "CALL" or "CANCEL" identifier "CALLDN (ALPHANUMERIC-EDITED)" was not alphanumeric, zoned decimal nor alphabetic. The statement was discarded.

Under IBM COBOL CMPR2, the above program will compile cleanly as in VS COBOL II Release 2. To get the same results under NOCMPR2, you can change the definition of CALLDN to all alphabetic or alphanumeric or add a new data-name redefined to CALLDN with a valid data type as shown below.

```
01 CALLDN PIC A(7).
        or
01 CALLDN PIC X(7).
        or

01 CALLDN PIC AAAAABB
01 CALLDN1 REDEFINES CALLDN PIC A(7).

    MOVE "PROG1" TO CALLDN1.
    CALL CALLDN1.
    CANCEL CALLDN1.
```

PROGRAM COLLATING SEQUENCE

CMPR2

The PROGRAM COLLATING SEQUENCE established in the OBJECT COMPUTER paragraph is used to determine the truth value of any nonnumeric comparisons that are:

- Explicitly specified in relation conditions
- Explicitly specified in condition-name conditions

- Implicitly performed as part of the execution of the SORT and MERGE statements, unless overridden by the COLLATING SEQUENCE phrase on the respective SORT or MERGE statement
- Implicitly performed as part of the execution of STRING, UNSTRING, and INSPECT statements

NOCMPR2

The PROGRAM COLLATING SEQUENCE established in the OBJECT COMPUTER paragraph is used to determine the truth value of any nonnumeric comparisons that are:

- Explicitly specified in relation conditions
- Explicitly specified in condition-name conditions
- Implicitly performed as part of the execution of the SORT and MERGE statements, unless overridden by the COLLATING SEQUENCE phrase on the respective SORT or MERGE statement

The **native collating sequence** is used to determine the truth value of any nonnumeric comparisons that are implicitly performed as part of the execution of STRING, UNSTRING, and INSPECT statements.

For most applications, this difference will not affect the results of these statements. Note that the implicit comparisons performed as part of STRING, UNSTRING, and INSPECT statements are always for equality. Therefore, even if the ordering of the characters in the PROGRAM COLLATING SEQUENCE is different than that of the native sequence, the results of these comparisons will be the same.

For an application to be affected by this change, the PROGRAM COLLATING SEQUENCE established in the OBJECT COMPUTER paragraph must identify an alphabet that was defined with the ALSO clause, which assigns several different characters to the same ordinal position.

Messages

Compiling the program with the CMPR2 and FLAGMIG options will cause the compiler to issue messages for all statements that may be affected by this change:

IGYPS3142-W **MIGR** The "PROGRAM COLLATING SEQUENCE" will not affect the "STRING" statement under the "NOCMPR2" compiler option.

Corrective action

No correction that is generally applicable exists for programs receiving this message if the PROGRAM COLLATING SEQUENCE contains multiple characters assigned to the same ordinal position.

The CMPR2 behavior of affected programs cannot be preserved without extensive and complex recoding. Such programs should be rewritten to avoid this dependency on the CMPR2 behavior.

READ INTO and RETURN INTO

READ (or RETURN) with the INTO phrase may be performed using a different record description if the file contains multiple record descriptions and not all of the record descriptions are alphanumeric.

A file will be affected only when it is fixed format, there are multiple 01 record descriptions, and at least one of the record descriptions is a numeric or numeric-edited item.

When deciding which record description to use as the sending field for the implicit MOVE, the compiler selects the longest of the 01 record descriptions. If multiple record descriptions have the same length, then the first such record description is chosen. This is true under both CMPR2 and NOCMPR2. However, the method for determining which 01 record description is the longest is different.

CMPR2

Under CMPR2, the length of numeric and numeric-edited record descriptions is calculated by totaling the number of digit positions in the PICTURE. Other types of record descriptions are assigned a length equal to the number of bytes occupied by the record description.

NOCMPR2

Under NOCMPR2, the length of each record description is determined to be the number of bytes occupied by the record description, regardless of whether the record description is numeric, numeric-edited, or otherwise.

Messages

If the FLAGMIG and CMPR2 compiler options are used, a message will be issued for any READ INTO or RETURN INTO statement that may be affected.

A program that is affected by the rule change will receive the following message:

IGYPS2281-I The "INTO" phrase of the "READ" or "RETURN" statement was specified for fixed-format file "file-name", which contained multiple records. Record "record-name" was selected as the sending field for the move.

This message will be issued under both the CMPR2 and NOCMPR2 compiler options. Therefore, you can compile the program with CMPR2, and then with NOCMPR2, and examine the messages to determine whether the same record was chosen under both CMPR2 and NOCMPR2. If so, then the program need not be changed.

In addition, with the FLAGMIG compiler option, the following message will appear:

IGYPS2283-W **MIGR** The "INTO" phrase of the "READ" or "RETURN" statement was specified for file "file-name", which contained multiple records. A different record may be selected for the sending field for the move under the "NOCMPR2" compiler option.

Corrective action

By applying the record description rules to each qualified file or by checking the messages, you can determine whether a different record description may be selected under NOCMPR2 than under CMPR2. For example, consider the following record descriptions:

```
01 RECORD-1 PIC X(9) USAGE DISPLAY.
01 RECORD-2 PIC 9(9) USAGE DISPLAY.
```

In this case, each record description is calculated to have a length of "9", both under CMPR2 and NOCMPR2. Therefore, no incompatibility exists.

Suppose, however, that there is a difference in the way that the record description lengths are calculated. Consider the following:

```
01 RECORD-3 PIC X(4) USAGE DISPLAY.
01 RECORD-4 PIC 9(9) USAGE COMP.
```

In this case, under NOCMPR2, each record description is calculated to have a length of "4". However, under CMPR2, the length of the numeric record description (RECORD-4) is calculated by counting digits, so its length will be "9" instead of "4". Thus, RECORD-4 will be used as the sending field, even though the byte length of each record description is 4.

Once an incompatibility has been detected, change the code to ensure that the CMPR2 behavior will be preserved. You can change the READ INTO or RETURN INTO statement to a READ or RETURN statement, followed by a MOVE statement. The MOVE statement would specify, as a sending field, the desired record description (the "longest" one), and, as a receiving field, the item that had been specified as the INTO item.

RECORD CONTAINS n CHARACTERS

The definition of RECORD CONTAINS n CHARACTERS affects existing programs and its behavior is different under CMPR2.

Consider the following example:

```
FD FILE1
   RECORD CONTAINS 40.
 01 F1R1 PIC X(20).
 01 F1R2 PIC X(40).


FD FILE2
   RECORD CONTAINS 20 TO 40.
 01 F2R1 PIC X(20).
 01 F2R2 PIC X(40).
```

CMPR2

Under CMPR2, FILE1 and FILE2 have variable-length records.

NOCMPR2

Under NOCMPR2, FILE1 has fixed-length records and FILE2 has variable-length records.

Message

Compiling the program with the CMPR2 and FLAGMIG options will cause the compiler to issue the following message for FILE1:

IGYPS1183-W **MIGR** "RECORD CONTAINS" clause with one integer specified is supported differently under the "NOCMPR2" compiler option.

Therefore, files created using VS COBOL II Release 2 (or IBM COBOL CMPR2) might not be processable using IBM COBOL NOCMPR2. File status 39 may result on OPEN.

Corrective action

To maintain current behavior, remove the RECORD CONTAINS clauses. This will result in both FILE1 and FILE2 having variable-length records.

For maximum clarity, and for any new applications, use RECORD CONTAINS n CHARACTERS for fixed-length records and RECORD IS VARYING FROM integer-1 TO integer-2 for variable-length records. Avoid using RECORD CONTAINS n1 TO n2 CHARACTERS.

Reserved words

CMPR2

The reserved words are the same as in VS COBOL II Release 2.

NOCMPR2

New reserved words were added for VS COBOL II Release 3, COBOL/370, and COBOL for MVS & VM. This section lists the words added for each product. The reserved word list is the same for the COBOL for MVS & VM and IBM COBOL compilers.

Note, the words added in any given release are also reserved in any subsequent releases.

Reserved words introduced in VS COBOL II Release 3

ALPHABET
ALPHABETIC-LOWER
ALPHABETIC-UPPER
BINARY
CLASS
COMMON
CONVERTING
DAY-OF-WEEK
DBCS
END-RECEIVE
EXTERNAL
GLOBAL
ORDER
PACKED-DECIMAL
PADDING
PURGE
REPLACE
STANDARD-2

Reserved words introduced in COBOL/370

FUNCTION
PROCEDURE-POINTER

Reserved words introduced in COBOL for MVS & VM

CLASS-ID
END-INVOKE
INHERITS
INVOKE
LOCAL-STORAGE
METACLASS
METHOD
METHOD-ID
OBJECT
OVERRIDE
RECURSIVE
REPOSITORY
RETURNING
SELF
SUPER

Messages

Compiling the program with CMPR2 and FLAGMIG compiler options will cause the compiler to issue a message for each new reserved word.

The following is an example for the new reserved word "ALPHABET":

IGYPS0057-W **MIGR** "ALPHABET" is a new COBOL reserved word under the "NOCMPR2" compiler option.

Corrective action

Change the reserved word to a different, nonreserved COBOL word by adding or subtracting characters, or change the words entirely.

SET...TO TRUE

CMPR2

The SET...TO TRUE statement is performed according to the rules of the MOVE statement.

NOCMPR2

Under NOCMPR2, SET...TO TRUE follows the rules of the VALUE clause. There are three instances in which this can cause different results:

- When the data item is described by a JUSTIFIED clause
- When the data item is described by a BLANK WHEN ZERO clause
- When the data item has editing symbols in its PICTURE string

Message

A program that is potentially affected by this change will receive the following message when compiled with the CMPR2 and FLAGMIG options:

IGYPS2219-W **MIGR** The "SET" statement with the "TO TRUE" phrase will be performed according to the rules for the "VALUE" clause under the "NOCMPR2" compiler option.

Corrective action

JUSTIFIED clause

When a data item described by a JUSTIFIED clause is the receiving item in a MOVE statement, the sending data is aligned at the rightmost character position in the receiving item. In a VALUE clause, initialization is not affected by the JUSTIFIED clause. This means that the data in a VALUE clause will be aligned at the leftmost character position in the receiving item.

Here's how it works under CMPR2:

```
01 A PIC X(3) JUSTIFIED RIGHT VALUE "a". (Result = "a  ")
   88 V VALUE "a".

SET V TO TRUE                            (Result = "  a")
MOVE "a" TO A                            (Result = "  a")
```

Here's how it works under NOCMPR2:

```
01 A PIC X(3) JUSTIFIED RIGHT VALUE "a". (Result = "a  ")
   88 V VALUE "a".
SET V TO TRUE                            (Result = "a  ")

MOVE "a" TO A                            (Result = "  a")
```

If using NOCMPR2, and you want the same behavior as with CMPR2, adjust the data in the VALUE clause for the 88-level item accordingly:

```
01 A PIC X(3) JUSTIFIED RIGHT VALUE "a". (Result = "a  ")
   88 V VALUE "  a".

SET V TO TRUE                            (Result = "  a")
MOVE "a" TO A                            (Result = "  a")
```

BLANK WHEN ZERO clause

When a data item described by a BLANK WHEN ZERO clause receives the value of zero in a MOVE statement, the item will contain nothing but spaces. In a VALUE clause, initialization is not affected by the BLANK WHEN ZERO clause. This means that if the VALUE clause specifies a value of zero, the data will be placed into the item as is, and the item will contain all zeros instead of spaces.

Here's how it works under CMPR2:

```
01 N PIC 9(3) BLANK WHEN ZERO VALUE ZERO. (Result = "000")
   88 V VALUE ZERO.

SET V TO TRUE                             (Result = "   ")
MOVE ZERO TO N                            (Result = "   ")
```

Here's how it works under NOCMPR2:

```
01 N PIC 9(3) BLANK WHEN ZERO VALUE ZERO. (Result = "000")
   88 V VALUE ZERO.
SET V TO TRUE                             (Result = "000")

MOVE ZERO TO N                            (Result = "   ")
```

If the behavior exhibited under CMPR2 is desired under NOCMPR2, the data in the VALUE clause for the 88-level item must be adjusted accordingly:

```
01 N PIC 9(3) BLANK WHEN ZERO VALUE ZERO. (Result = "000")
   88 V VALUE "   ".

SET V TO TRUE                             (Result = "   ")
MOVE ZERO TO N                            (Result = "   ")
```

PICTURE string with editing symbols

When a data item contains editing symbols in its PICTURE string, the character positions represented by those symbols will contain editing characters when data is moved into the data item. In a VALUE clause, initialization is not affected by the editing symbols. This means that the

data in the VALUE clause will be placed into the item as is, and editing will not take place as it does in the MOVE statement.

Here's how it works under CMPR2:

```
01 E PIC X/X VALUE SPACE. (Result = "   ")
   88 V VALUE SPACE.

SET V TO TRUE             (Result = " / ")
MOVE SPACE TO E           (Result = " / ")
```

Here's how it works under NOCMPR2:

```
01 E PIC X/X VALUE SPACE. (Result = "   ")
   88 V VALUE SPACE.
SET V TO TRUE             (Result = "   ")

MOVE SPACE TO E           (Result = " / ")
```

If the behavior exhibited under CMPR2 is desired under NOCMPR2, the data in the VALUE clause for the 88-level item must be specified in edited form:

```
01 E PIC X/X VALUE SPACE. (Result = "   ")
   88 V VALUE " / ".

SET V TO TRUE             (Result = " / ")
MOVE SPACE TO E           (Result = " / ")
```

SIZE ERROR on MULTIPLY and DIVIDE

The COBOL Standards ('74 and '85) state that an intermediate result will be provided by the implementer when a COMPUTE, DIVIDE, or MULTIPLY statement has multiple receiving fields. For example: `MULTIPLY A BY B GIVING C D` should behave like:

```
MULTIPLY A BY B GIVING temp
MOVE temp TO C
MOVE temp TO D
```

where *temp* is an intermediate result provided by the implementer.

The *IBM COBOL Programming Guide* (for the version of IBM COBOL you are using) describes the use and definition of intermediate results. One such definition says that an intermediate result will have at most 30 digits.

So, in the example above, if A, B, C, and D are all defined as PIC S9(18), A will be multiplied by B, yielding a 36-digit result, which will be moved to the 30-digit intermediate result, *temp*. Then *temp* will be moved to C and D.

CMPR2

When SIZE ERROR is specified on the MULTIPLY statement example, SIZE ERROR can occur when the 36-digit (immediate) result is moved into the 30-digit (intermediate) result, according to the COBOL 74 Standard rules. This differs from the corresponding COMPUTE case, in which SIZE ERROR cannot occur when the 36-digit (immediate) result is moved into the 30-digit (intermediate) result.

```
COMPUTE C D = A * B ON SIZE ERROR...
```

This applies to the DIVIDE statement with its corresponding COMPUTE statement as well.

NOCMPR2

However, under NOCMPR2, SIZE ERROR applies only to final results. In the MULTIPLY example, SIZE ERROR cannot occur when the 36-digit (immediate) result is moved into the 30-digit (intermediate) result. Consequently, the MULTIPLY and COMPUTE statements become equivalent in this regard. This also applies to the DIVIDE statement.

Such statements will now be flagged by the following compiler message:

IGYPG3113-W Truncation of high-order digit positions may occur due to precision of intermediate results exceeding 30 digits.

If, at run time, truncation actually does occur, the following message will appear:

IGZ0036W Truncation of high order digit positions occurred in program "program-name" on line number "n".

Message

A program that is potentially affected by this change will receive the following message when compiled with the CMPR2 and the FLAGMIG options:

IGYPG3204-W **MIGR** The "ON SIZE ERROR" phrase will not be executed for intermediate results under the "NOCMPR2" compiler option.

Corrective action

The CMPR2 behavior of affected programs cannot be preserved without extensive and complex recoding. Such programs should be rewritten to avoid this dependency on the CMPR2 behavior.

UNSTRING operand evaluation

In the description below, the following general format of the UNSTRING statement is used for reference:

```
UNSTRING id-1
  DELIMITED BY id-2 OR id-3 ...
  INTO id-4 DELIMITER IN id-5 COUNT IN id-6
       id-7 DELIMITER IN id-8 COUNT IN id-9
  ⋮
  WITH POINTER id-10
  TALLYING IN id-11
  ON OVERFLOW imp-stmt-1
  NOT ON OVERFLOW imp-stmt-2
  END-UNSTRING
```

CMPR2

Under CMPR2, any subscripting, indexing, or length calculation associated with id-1, id-10, and id-11 is to be evaluated only once, at the beginning of execution of the UNSTRING statement. However, any subscripting, indexing, or length calculation associated with id-2, id-3, id-4, id-5, id-6, id-7, id-8, and id-9, (or any repetitions) is to be evaluated immediately before transfer into the respective data item.

NOCMPR2

Under NOCMPR2, any subscripting, indexing, or length calculation associated with any of id-1 through id-11 (or any repetitions) is to be evaluated only once, at the beginning of execution of the UNSTRING statement. This can lead to different results when certain dependencies exist between id-2 through id-9.

Note: Dependencies involving identifiers id-1, id-10, and id-11 are not affected by this change.

Messages

Most of the UNSTRING statements flagged with messages 3211 through 3214 will generate identical results. Only certain dependencies between the operands in the UNSTRING statement will generate different results.

For example, a dependency can exist between two operands (op-1 and op-2) in an UNSTRING statement in the following ways:

1. op-1 is subscripted, and the subscript value is modified by op-2:
   a. The subscript identifier is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.
   b. The subscript identifier is a variably-located item, and an ODO object affecting the location of this item is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.
2. op-1 is a variable-length group item, and an ODO object affecting the length of this group is modified by op-2:
   a. The ODO object is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.
3. op-1 is a variably-located item, and an ODO object affecting the location of this item is modified by op-2:
   a. The ODO object is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.

Note: Dependencies generated by overlapping operands, or by specifying the same identifier as a DELIMITED BY operand and as one of the sending, INTO, or DELIMITER IN operands are illegal under both the COBOL 74 and 85 Standards and are not addressed here. Generally, results will be unpredictable.

Compiling the program under IBM COBOL (with the CMPR2 and FLAGMIG options) will cause the compiler to issue messages for all UNSTRING statements that may contain such dependencies.

Any UNSTRING statements not flagged with one of these messages will generate identical results under VS COBOL II Release 2 CMPR2, IBM COBOL NOCMPR2.

All UNSTRING statements flagged with message 2222 will require changes to guarantee identical results.

Corrective action

The individual cases requiring changes are detailed below in order by message number, and with examples illustrating the dependencies and the suggested changes. Only the essential program fragments are included in the examples.

IGYPS2222-W

This message will be issued if one of the "receiver" identifiers in the UNSTRING statement refers to a variable-length group item that contains its own ODO object. Due to the syntax rules and restrictions applying to all UNSTRING statements, this can occur only for id-2, id-3, id-4, id-5, id-7, and id-8 (or repetitions).

For example:

```
01 VLG-1.
 02 VLG-1-ODOOBJ PIC 9 VALUE IS 5.
 02 VLG-1-GR.
  03 VLG-1-ODO PIC X OCCURS 1 TO 9 TIMES
                     DEPENDING ON VLG-1-ODOOBJ.
77 S-1 PIC X(20) VALUE IS ALL "123456789".

   UNSTRING S-1
            INTO VLG-1
            END-UNSTRING
```

IGYPS2222-W **MIGR** The maximum length of receiver "vlg-1" will be used under the "NOCMPR2" compiler option.

Note: IBM COBOL NOCMPR2 will use the maximum length of vlg-1 to determine both the amount of data extracted from sending item s-1 and the length of the receiving area vlg-1.

Regardless of which identifier is flagged with message 2222, you must replace the identifier with a reference modified version, as in the following:

```
UNSTRING S-1
         INTO VLG-1(1:LENGTH OF VLG-1)
         END-UNSTRING
```

This form forces the actual length of vlg-1 at the beginning of the UNSTRING statement to be used.

This correction is not affected by the presence of any of the optional phrases of the UNSTRING statement (DELIMITED BY, WITH POINTER, ON OVERFLOW) and it applies equally to all flagged identifiers in any one UNSTRING statement.

IGYPA3211-W

This message will be issued if one of the "DELIMITED BY" identifiers in the UNSTRING statement is subscripted, refers to a variable-length group item, or refers to a variably-located item.

For an UNSTRING statement to be affected by this change, the flagged DELIMITED BY operand must be dependent on one of the INTO receivers.

For example:

```
01 DEL.
 02 OCC-DEL-1 PIC X OCCURS 9 TIMES.
 02 VLEN-DEL-2-ODOOBJ PIC 9 VALUE IS 5.
 02 VLEN-DEL-2.
  03 VLEN-DEL-2-ODO PIC X OCCURS 1 TO 9 TIMES
                         DEPENDING ON VLEN-DEL-2-ODOOBJ.

77 S-1 PIC X(20) VALUE IS ALL "123456789".
77 R-1 PIC X(20) VALUE IS SPACES.
77 R-2 PIC X(20) VALUE IS SPACES.
77 SUB-5 PIC 99 VALUE IS 5.

   UNSTRING S-1
            DELIMITED BY OCC-DEL-1(SUB-5) OR VLEN-DEL-2,
            INTO R-1 DELIMITER IN OCC-DEL-1(SUB-5 + 1)
                     COUNT IN VLEN-DEL-2-ODOOBJ,
                 R-2,
            END-UNSTRING
```

IGYPA3211-W **MIGR** In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "DELIMITED BY" operand will be done only once under the "NOCMPR2" compiler option.

No corrections are required for items flagged with message 3211.

IGYPA3212-W

This message will be issued if one of the INTO identifiers in the UNSTRING statement is subscripted, refers to a variable-length group item, or refers to a variably-located item.

If an UNSTRING statement is to be affected by this change, the flagged INTO identifier must be dependent on one of the receivers in a preceding INTO phrase.

For example:

```
01 REC.
 02 R-1 PIC X(20) VALUE IS SPACES.
 02 R-2-SUB PIC 9 VALUE IS 9.
 02 OCC-R-2-GR.
  03 OCC-R-2 PIC X OCCURS 9 TIMES.
 02 R-3-ODOOBJ PIC 9 VALUE IS 9.
 02 ODO-R-3.
  03 FILLER PIC X OCCURS 1 TO 9 TIMES
                 DEPENDING ON R-3-ODOOBJ.

77 S-3 PIC X(20) VALUE IS "12 345  6789   .....".

   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN R-2-SUB,
                 OCC-R-2(R-2-SUB) COUNT IN R-3-ODOOBJ,
                 ODO-R-3,
            END-UNSTRING
```

IGYPA3212-W **MIGR** In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "INTO" operand will be done only once under the "NOCMPR2" compiler option.

This UNSTRING statement will generate different results under CMPR2 and NOCMPR2 because the subscript of the second INTO receiver is modified by the COUNT IN receiver of the first INTO phrase. In addition, the length of the third INTO receiver is modified by the COUNT IN receiver of the second INTO phrase.

Under CMPR2, the values moved to the COUNT IN identifiers will be used for the subsequent INTO phrases. Under NOCMPR2, the values in effect at the beginning of the execution of the UNSTRING statement will be used for all INTO phrases.

Any UNSTRING statement flagged with message 3212 must be broken into multiple UNSTRING statements. A separate UNSTRING statement must be used for each dependent INTO phrase. However, be aware of the following:

- If the original UNSTRING statement specified a WITH POINTER phrase, that phrase must be included in all of the changed UNSTRING statements. If the original UNSTRING statement **did not** specify a WITH POINTER phrase, that phrase must be added to all the changed UNSTRING statements, and the POINTER identifier must be initialized to 1.
- If the original UNSTRING statement specified a TALLYING IN phrase, that phrase must be included in all of the changed UNSTRING statements.
- If the original UNSTRING statement specified the ON OVERFLOW or NOT ON OVERFLOW phrases, those phrases must be included only in the last of the changed UNSTRING statements.

With these changes, the previous example becomes:

```
77 PTR PIC 99.

   MOVE 1 TO PTR
   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN R-2-SUB,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO OCC-R-2(R-2-SUB) COUNT IN R-3-ODOOBJ,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO ODO-R-3,
            WITH POINTER PTR,
            END-UNSTRING
```

IGYPA3213-W

This message will be issued if one of the DELIMITER IN identifiers in the UNSTRING statement:

- Is subscripted
- Refers to a variable-length group item
- Refers to a variably-located item

If an UNSTRING statement is to be affected by this change, the flagged DELIMITER IN identifier must be dependent on one of the receivers in a preceding INTO phrase.

Note: Dependencies between identifiers in the same INTO phrase will not affect the result of the UNSTRING statement. CMPR2 behavior delays the effects of these dependencies until the next INTO phrase.

For example:

```
01 DEL.
 02 D-2-SUB PIC 9 VALUE IS 9.
 02 OCC-D-2-GR.
  03 OCC-D-2 PIC X OCCURS 9 TIMES.
 02 D-3-ODOOBJ PIC 9 VALUE IS 9.
 02 ODO-D-3.
  03 FILLER PIC X OCCURS 1 TO 9 TIMES
                DEPENDING ON D-3-ODOOBJ.

77 S-4 PIC X(20) VALUE IS "12 345  6789   .....".
77 R-1 PIC X(20) VALUE IS SPACES.
77 R-2 PIC X(20) VALUE IS SPACES.
77 R-3 PIC X(20) VALUE IS SPACES.

   UNSTRING S-4
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN D-2-SUB,
                 R-2 DELIMITER IN OCC-D-2(D-2-SUB)
                     COUNT IN D-3-ODOOBJ,
                 R-3 DELIMITER IN ODO-D-3,
            END-UNSTRING
```

IGYPA3213-W **MIGR** In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "DELIMITER IN" operand will be done only once under the "NOCMPR2" compiler option.

This UNSTRING statement will generate different results under CMPR2 and NOCMPR2 because the subscript of the DELIMITER IN identifier of the second INTO phrase is modified by the COUNT IN receiver of the first INTO phrase. In addition, the length of the DELIMITER IN identifier of the third INTO phrase is modified by the COUNT IN receiver of the second INTO phrase.

With CMPR2 behavior, the values moved to the COUNT IN identifiers will be used for the subsequent INTO phrases. With NOCMPR2, the values in effect at the beginning of the execution of the UNSTRING statement will be used for all INTO phrases.

Any UNSTRING statement flagged with message 3213 must be broken into multiple UNSTRING statements; a separate UNSTRING statement must be used for each dependent INTO phrase.

With these changes, the previous example becomes:

```
77 PTR PIC 99.
   MOVE 1 TO PTR
   UNSTRING S-4
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN D-2-SUB,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-4
            DELIMITED BY ALL SPACES,
            INTO R-2 DELIMITER IN OCC-D-2(D-2-SUB)
                     COUNT IN D-3-ODOOBJ,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-4
            DELIMITED BY ALL SPACES,
            INTO R-3 DELIMITER IN ODO-D-3,
            WITH POINTER PTR,
            END-UNSTRING
```

IGYPA3214-W

This message will be issued if one of the COUNT IN identifiers in the UNSTRING statement is subscripted, or refers to a variably-located item.

In order for an UNSTRING statement to be affected by this change, the flagged COUNT IN identifier must be dependent on one of the receivers in a preceding INTO phrase.

Note: Dependencies between identifiers in the same INTO phrase will not affect the result of the UNSTRING statement; CMPR2 behavior delays the effects of these dependencies to the next INTO phrase.

For example:

```
01 C-2.
 02 C-2-SUB PIC 9 VALUE IS 9.
 02 OCC-C-2-GR.
  03 OCC-C-2 PIC 9 OCCURS 9 TIMES.

77 S-5 PIC X(20) VALUE IS "12 345  6789........".
77 R-1 PIC X(20) VALUE IS SPACES.
77 R-2 PIC X(20) VALUE IS SPACES.

   UNSTRING S-5
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN C-2-SUB,
                 R-2 COUNT IN OCC-C-2(C-2-SUB),
            END-UNSTRING
```

IGYPA3214-W **MIGR** In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "COUNT IN" operand will be done only once under the "NOCMPR2" compiler option.

This UNSTRING statement will generate different results under CMPR2 and NOCMPR2 because the subscript of the COUNT IN identifier of the second INTO phrase is modified by the COUNT IN receiver of the first INTO phrase.

With CMPR2 behavior, value moved to the COUNT IN identifier in the first INTO phrase will be used for the second INTO phrase. With NOCMPR2, the value in effect at the beginning of execution of the UNSTRING statement will be used.

Any UNSTRING statement flagged with message 3214 must be broken into multiple UNSTRING statements; a separate UNSTRING statement must be used for each dependent INTO phrase.

With these changes, the above example becomes:

```
77 PTR PIC 99.

   MOVE 1 TO PTR
   UNSTRING S-5
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN C-2-SUB,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-5
            DELIMITED BY ALL SPACES,
            INTO R-2 COUNT IN OCC-C-2(C-2-SUB),
            WITH POINTER PTR,
            END-UNSTRING
```

UPSI switches

CMPR2

UPSI switches may be defined by specifying condition-names for the ON and OFF settings of the switch. Under CMPR2, the condition-names for all UPSI switches, UPSI-0 through UPSI-7, can be defined with the same names, as follows:

```
SPECIAL-NAMES.
    UPSI-0  ON STATUS IS T  OFF STATUS IS F
    UPSI-1  ON STATUS IS T  OFF STATUS IS F
  ⋮
    UPSI-7  ON STATUS IS T  OFF STATUS IS F
```

References to the names could be qualified with the UPSI name, as follows:

```
IF T OF UPSI-0 DISPLAY "UPSI-0".
IF T OF UPSI-1 DISPLAY "UPSI-1".
  ⋮
IF T OF UPSI-7 DISPLAY "UPSI-7".
```

NOCMPR2

The names of the UPSI switches, UPSI-0 through UPSI-7, can no longer be referenced in the Procedure Division under NOCMPR2. The above statements will now be flagged with a message of the following format:

IGYPS2121-S "T OF UPSI-0" was not defined as a data-name. The statement was discarded.

Message

Using CMPR2 and FLAGMIG, any Procedure Division statement that references an UPSI switch by name will be flagged with the following message:

IGYPS0186-W **MIGR** UPSI switches cannot be referenced directly in the Procedure Division under the "NOCMPR2" compiler option.

Corrective action

Programs will have to be changed to define unique condition-names, as follows:

```
SPECIAL-NAMES.
  UPSI-0  ON STATUS IS T0  OFF STATUS IS F0
  UPSI-1  ON STATUS IS T1  OFF STATUS IS F1
  ⋮
  UPSI-7  ON STATUS IS T7  OFF STATUS IS F7
```

and to reference the new condition-names, as follows:

```
IF T0 DISPLAY "UPSI-0".
IF T1 DISPLAY "UPSI-1".
  ⋮
IF T7 DISPLAY "UPSI-7".
```

Variable-length group moves

CMPR2

All ODO objects in sending and receiving fields involved in a group move, such as a MOVE statement, must be set before the statement is executed. The actual lengths of the sender and receiver are calculated just before the execution of the data movement statement. For a list of affected verbs, see "Message" below.

NOCMPR2

In some cases, NOCMPR2 uses the maximum length of a variable-length group when it is a receiver, whereas CMPR2 uses the actual length. This occurs when the receiver is variable length, contains its own ODO object, and is the last group in a structure. For example:

```
01  ODO-SENDER.
    02  SEND-OBJ PIC 99.
    02  SEND-ITEM PIC X OCCURS 1 TO 20 DEPENDING ON SEND-OBJ.

01  ODO-RECEIVER.
    02  RECV-OBJ PIC 99.
    02  RECV-ITEM PIC X OCCURS 1 TO 20 DEPENDING ON RECV-OBJ.
  ⋮
MOVE 5 TO SEND-OBJ.
MOVE 10 TO RECV-OBJ.
MOVE ODO-SENDER TO ODO-RECEIVER.
  ⋮
CMPR2:
   Occurrences 1-5 of ODO-SENDER moved to ODO-RECEIVER.
   Occurrences 6-10 of ODO-RECEIVER become spaces.
   Occurrences 11-20 of ODO-RECEIVER are unchanged.
NOCMPR2:
   Occurrences 1-5 of ODO-SENDER moved to ODO-RECEIVER.
   Occurrences 6-20 of ODO-RECEIVER become spaces.
```

The programs that will have negative effects if used under NOCMPR2 are those that reference occurrences of the table that are beyond the value of the ODO object at the time a data movement statement was executed.

In the example above, if occurrences 11-20 have data in them before the group move, that data will be lost after the group move if run under NOCMPR2.

Message

Compiling the program with the CMPR2 and FLAGMIG compiler options will generate the following message for each data movement statement that will behave differently under NOCMPR2:

IGYPS2222-W **MIGR** The maximum length of receiver "ODO-RECEIVER" will be used under the "NOCMPR2" compiler option.

This difference in variable-length group moves affects any verb that moves data. The list of affected verbs follows:

ACCEPT identifier (Format 1 or Format 2)
MOVE ... TO identifier
READ ... INTO identifier
RELEASE identifier FROM ...
RETURN ... INTO identifier
REWRITE identifier FROM ...
STRING ... INTO identifier
UNSTRING ... INTO identifier DELIMITER IN identifier
WRITE identifier FROM ...

Corrective action

Step 1 See if any of your COBOL programs have the variable-length data movement statements by compiling them under IBM COBOL with the CMPR2 and FLAGMIG compiler options. This will flag all variable-length group moves with receivers that contain their own ODO objects and are not complex ODO items.

Step 2 See if any data that was previously left unchanged and is now being set to blanks is referenced after the data movement statements. In the example, if the ODO object has a value of 5 and a maximum value of 10 and the code uses data in occurrence numbers 6 through 10 after the MOVE, then the program will have different results between CMPR2 and NOCMPR2.

Step 3 Change the receiver in the data movement statement to use reference modification to specify explicitly the length of the receiving field. For example:

```
MOVE ODO-SENDER TO ODO-RECEIVER (1:LENGTH OF ODO-RECEIVER).
```

Appendix M. IMS considerations

When running any combination of OS/VS COBOL programs, VS COBOL II programs, and IBM COBOL programs under Language Environment, you need to know:

- Unsupported features from VS COBOL II
- Compiler options relevant for programs run on IMS
- ENDJOB/NOENDJOB compiler option requirements
- Recommended modules for preload
- Condition handling using CBLTDI on IMS
- Performance consideration when running OS/VS COBOL programs
- The use of a GTF trace to determine which modules are loaded
- DFSPCC20 modification is unsupported

Unsupported VS COBOL II features

BLDL user exit unsupported: The VS COBOL II BLDL user exit is not supported in Language Environment. No replacement is available. However, Language Environment (starting with Version 1 Release 8) does have a load notification exit that might meet your needs. For information on the load notification exit, see *Language Environment for OS/390 Customization*.

LIBKEEP unsupported: The VS COBOL II LIBKEEP run-time option is not supported in Language Environment. For alternatives to LIBKEEP, see "Existing applications using LIBKEEP" on page 92.

Compiler options relevant for programs run on IMS

Figure 60 lists IBM COBOL compiler options relevant to COBOL IMS applications.

Figure 60. Compiler options relevant for IBM COBOL programs run on IMS

Compiler option	Comments
RENT	The RENT compiler option causes IBM COBOL to produce reentrant code and allows you to place the COBOL modules in the MVS Link Pack Area or Extended Link Pack Area and thus shared among dependent regions under IMS. Also, the modules cannot be overwritten, since the LPA/ELPA have a storage protect key. RENT allows you to run your IMS programs in either preload or nonpreload mode, without compiling with different options.

ENDJOB/NOENDJOB compiler option requirements

If you are running a mixture of OS/VS COBOL and IBM COBOL programs, at least one of the OS/VS COBOL programs must have been compiled with the ENDJOB option if the programs are not preloaded.

Specifying the ENDJOB option allows the COBOL application and the Language Environment run-time environment to be completely cleaned up at program termination.

Note: If LRR is used at the same time as OS/VS COBOL and IBM COBOL, the run-time environment will not be completely cleaned up, because LRR forces NOENDJOB behavior.

Preloading requirements

If you preload ILBOCOM, you must also preload the following library routines:

- ILBOCMM, ILBONTR, and ILBOSRV
- ILBOACS, ILBOCVB, and ILBOINS if an OS/VS COBOL RES program uses the INSPECT and/or UNSTRING verbs.

Last used state behavior under Language Environment

The OS/VS COBOL NOENDJOB/ENDJOB compiler option produces different results when running with the Language Environment library than when running with the OS/VS COBOL library. When specifying NOENDJOB and running under OS/VS COBOL, your subprograms are always entered in the last used state.

In the Language Environment run-time environment, when nonpreloaded OS/VS COBOL subprograms that are compiled with RES and NOENDJOB terminate, the programs and their internal work areas are deleted; their external work areas (storage acquired with a GETMAIN) and library routines remain in dynamic storage. When ENDJOB is used instead of NOENDJOB, there is essentially no difference except that the external work areas and library routines are deleted as well.

Because IBM COBOL programs and their resources (all work areas and library routines) are always deleted at termination, the way ENDJOB and NOENDJOB work in the Language Environment run-time environment allows both IBM COBOL and OS/VS COBOL programs to be treated similarly with respect to last-used state.

In the OS/VS COBOL run-time environment, ENDJOB was the strongly recommended compiler option when executing IMS transactions invoking nonpreloaded COBOL programs.

In the VS COBOL II run-time environment when using the LIBKEEP run-time option, OS/VS COBOL programs compiled with ENDJOB were treated as NOENDJOB. NOENDJOB was required for preloaded OS/VS COBOL programs.

Note: When using LRR, NOENDJOB behavior is always in effect.

When programs remain in the last-used state

A program remains in last-used state between COBOL transactions only when all the following conditions are true:

- It is an OS/VS COBOL program compiled with the NOENDJOB option or LRR is being used.
- It has been link-edited with the REUS option.
- It has been preloaded.
- It is a dynamically called subprogram or a statically called subprogram that has, itself, been called by a dynamically called subprogram.

Recommended modules for preload

IMS allows application programs to be preloaded, (to remain resident in storage after each use). Preloading can improve performance; if the program is already resident in storage, requests for the program can be handled faster. No time is wasted bringing it into storage from an external medium.

IBM COBOL programs: For a list of recommended modules to preload, see the *IBM COBOL for MVS & VM V1R2 Performance Tuning Paper*, located on the Web, in the Library Section, at: `http://www.ibm.com/software/ad/cobol`

Go to the Library section by clicking on the `Library` link.

OS/VS COBOL programs: Figure 61 shows a sample list of OS/VS COBOL compatibility subroutines to preload if you use the OS/VS COBOL RES compiler option. If you run OS/VS COBOL programs with Language Environment, you might also want to preload the modules listed in the *Performance Paper*.

ILBOCHN0	ILBOCMM0	ILBOCOM0
ILBOCVB0	ILBODTE0	ILBOETB0
ILBOGDO0	ILBOINS0	ILBOITB0
ILBONTR0	ILBOSCH0	ILBOUST0
ILBOWTB0	ILBOSRV0	ILBOSTG0
ILBOSTT2	ILBOTRN0	

Figure 61. Sample list for preloaded ILBO modules

When running under Language Environment:

- Continue to preload the same ILBO library routines as you preloaded when using the OS/VS COBOL library. (These ILBO routines are included in Language Environment.)
- Preload ILBOSTT2 (and put it in the preload list twice) **if** you preloaded ILBOSTT for your OS/VS COBOL programs running with the OS/VS COBOL library.

Condition handling using CBLTDLI on IMS

The CEETDLI and PLITDLI interfaces keep track of calls to and returns from IMS. If you use CBLTDLI or ASMTDLI from a COBOL program and you are using IMS Version 4 Release 1 or IMS Version 3 Release 1 with PTF UN49280, the coordination between Language Environment and IMS condition handling is the same as that provided by CEETDLI. To ensure compatibility, specify the Language Environment ABTERMENC(ABEND) and TRAP(ON) run-time options.

Differences with IMS Version 2 and Version 3

CBLTDLI or ASMTDLI (issued from a non-PL/I routine) running with IMS/ESA Version 2 Release 2 (or Version 3 Release 1 without PTF UN49280) under Language Environment do not keep track of calls to and returns from IMS.

Without the coordination of condition handling between IMS and Language Environment, if a program interrupt or abend occurs, the Language Environment condition manager is not informed whether the problem occurred in your application or in IMS.

To ensure that your database does not get contaminated if a condition occurs:

1. **Do not** handle the condition with condition handling. If you do attempt to handle the errors using condition handlers, the integrity of your IMS database is at risk.
2. Use the ABTERMENC(ABEND) and TRAP(ON) run-time options to ensure that the application terminates abnormally and transforms all abnormal terminations into operating system abends to cause IMS rollbacks.

Figure 62 shows you when you can use condition handlers with the various IMS releases.

Figure 62. Restrictions on condition handling under IMS (by release)

Interface	IMS V2	IMS V3 w/o PTF UN49280	IMS V3 with PTF UN49280	IMS V4 and later
CEETDLI	N/A	N/A	N/A	Yes
CBLTDLI	No	No	Yes	Yes
PLITDLI	Yes	Yes	Yes	Yes
CTDLI	Yes	Yes	Yes	Yes
ASMTDLI (non-PL/I)	No	No	Yes	Yes
ASMTDLI (PL/I)	Yes	Yes	Yes	Yes

Performance consideration when running OS/VS COBOL programs

If most of the programs scheduled in your IMS region are OS/VS COBOL programs, a performance benefit might be acheived by specifying lower values for the Language Environment storage related run-time options. For example, you could use the following storage values:

```
STACK(16K)
HEAP(4K,4K,ANY,4K,4K)
BELOWHEAP(4K)
ANYHEAP(4K)
```

Using a GTF trace to determine which modules are loaded

Some customers collect GTF trace information and look at the SVC 8 (LOAD) trace information to determine which application programs are loaded. Language Environment uses both SVC 8 and SVC 122 to load programs.

DFSPCC20 modification unsupported

OS/VS COBOL allowed the use of the ENDJOB option for both preloaded and nonpreloaded programs with a user modification to the IMS Program Controller (DFSPCC20).

The user modification included code that would store the addresses of certain preloaded subroutines and might cause intermittent problems at execution. This user

modification (or any user supplied code which performs the same task) must be removed before running under Language Environment.

Appendix N. CMS considerations

This appendix describes conversion considerations for programs running on CMS. It includes information on:

- Preserving storage under CMS
- Using the batch compile facility
- Placing dynamic CALLs to alternate entry points
- Using ISPF with CMS
- Changes in EXTERNAL storage
- Mixing static and dynamic CALLS
- Using load modules

Preserving storage under CMS

When you compile OS/VS COBOL programs with the OS/VS COBOL NOENDJOB compiler option and run them under VM/ESA, you must specify the CMS command "SET STORECLR=ENDCMD". This command ensures that storage will be preserved by CMS until you receive the CMS READY message. Thus, the EXECOS command can operate normally. If "STORECLR=ENDSVC" is in effect and an OS/VS COBOL program is compiled with the NOENDJOB option, results are unpredictable.

If you use EXECOS in conjunction with the SET STORECLR=ENDCMD, NOENDJOB compiler option has no effect. EXECOS deletes the environment left by NOENDJOB. Subsequent execution of an OS/VS COBOL program will reinitialize the environment and reload the OS/VS COBOL compatibility library routines.

Using the batch compile facility

Dynamic CALL literal, CALL identifier, or CANCEL statements that target a subprogram compiled into the same object program as the calling program (using the batch compile facility) are only supported if the CMS LKED command is used to generate independent load modules for each program in the object deck.

Placing dynamic CALLs to alternate entry points

With IBM COBOL, dynamic calls placed to an alternate entry point within a COBOL program will enter the COBOL program at the main entry point, **not** the alternate entry point. You must put the COBOL programs in a CMS LOADLIB to have dynamic calls enter at the alternate entry point.

You must also place your COBOL applications in a CMS LOADLIB if an assembler program uses SVC 8 to load a COBOL application that contains OS/VS COBOL and calls it multiple times.

Using ISPF with CMS

If you use ISPF, be aware of the changes in the CMS parameter list and split screen support.

CMS style parameter list

If you code your application using ISPF panels, you can gain interactive access to your COBOL application. When a COBOL application is invoked from ISPF, the parameters that are passed must be compatible with the CMS parameter style supported by Language Environment. To do this, specify the CMS option when you use the ISPF command ISPEXEC SELECT PGM to invoke a COBOL routine.

Split screen support

Under CMS, both run-time options and program arguments can be passed to the COBOL routine. The order in which these are interpreted by Language Environment depends upon the setting of CBLOPTS.

Changes in EXTERNAL storage—VS COBOL II only

When running Language Environment on CMS, use of CMSCALL will result in a creation of a new enclave. For example, comparing VS COBOL II and Language Environment, if program A (COBOL) CALLs program B (assembler), and program B does a CMSCALL to program C (COBOL):

1. With VS COBOL II EXTERNAL data is shared between program A and program C (both programs are in the same enclave).
2. With Language Environment, a new enclave is created. Now, EXTERNAL storage is **not** shared between program A and program C. Each program now has its own copy of the EXTERNAL data (since the scope of EXTERNAL data is at the enclave level).

Mixing static and dynamic CALLs

If a VS COBOL II or IBM COBOL program is statically linked or loaded with another VS COBOL II or IBM COBOL program, do not issue dynamic CALLs between the two programs. CMS rewrites the loader tables when running under Language Environment, which could cause the application to fail.

Using load modules

When you generate a load module for your application (using the CMS GENMOD command), you should use the NOMAP option on the GENMOD command to avoid the possibility of getting a duplicate identifier message from the CMS loader for csects in your load module that might also appear in other programs that your application uses.

Appendix O. TSO considerations

This appendix describes conversion considerations for programs running on TSO. It includes information on using REXX execs.

Using REXX execs

When you run a COBOL program from a REXX exec, you need to be aware of the differences in the parameter list formats for using the different "address" options. When using 'Address TSO' (the default) or 'Address ATTCHMVS', both program parameters and Language Environment run-time options are processed. When using 'Address LINKMVS', run-time options are not processed, but they are passed as program parameters to the COBOL program.

Due to the differences in parameter list formats and save area conventions, 'Address LINK', 'Address ATTACH', 'Address LINKPGM', and 'Address ATTCHPGM' are not supported.

Appendix P. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, HHX/H3
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Programming interface information

This book is intended to help you write programs using IBM COBOL for OS/390 & VM and IBM COBOL for MVS & VM. This *Compiler and Run-Time Migration Guide* documents General-Use Programming Interface and Associated Guidance Information provided by COBOL for OS/390 & VM and COBOL for MVS & VM.

General-Use programming interfaces allow the customer to write programs that obtain the services of COBOL for OS/390 & VM and COBOL for MVS & VM.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AD/Cycle
AIX
C/370
CICS
CICS/ESA
COBOL/370
DB2
DFSMS
DFSORT
IBM
IMS
IMS/ESA
Language Environment
MVS
MVS/ESA
OpenEdition
Operating System/2
OS/2
OS/390
S/370
SOMobjects
System Object Model
System/370
VisualAge
VM/ESA
VSE/ESA

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Bibliography

COBOL for OS/390 & VM

Compiler and Run-Time Migration Guide, GC26-4764

Customization under OS/390, GC26-9045

Debug Tool User's Guide and Reference, SC09-2137

Diagnosis Guide, GC26-9047

Fact Sheet, GC26-9048

Language Reference, SC26-9046

Licensed Program Specifications, GC26-9044

Programming Guide, SC26-9049

Language Environment for OS/390 & VM

- *Concepts Guide*, GC28-1945
- *Debugging Guide and Run-Time Messages* SC28-1942
- *Installation & Customization*, SC28-1941
- *Programming Guide*, SC28-1939
- *Programming Reference*, SC28-1940
- *Run-Time Migration Guide*, SC28-1944
- *Writing Interlanguage Applications*, SC28-1943

Other products

- **SOMobjects for MVS**

  Introducing SOMobjects on MVS, GC28-1529

  Licensed Program Specifications, GC28-1534

  User's Guide, GC28-1545

  Class Library Reference, SC28-1546

  Reference Summary, SC28-1547

- **IBM C/370**

  Migration Guide, SC09-1359

  Programming Guide, SC09-1356

  General Information, GC09-1358

  Diagnosis Guide, LY09-1806

  Licensed Program Specifications, GC09-1357

  Reference Summary, SX09-1247

- **IBM PL/I for MVS & VM**

  Fact Sheet, GC26-3112

  Licensed Program Specification, GC26-3116

  Compiler and Run-Time Migration Guide, SC26-3118

  Installation and Customization under MVS, SC26-3119

  Installation and Customization under CMS, SC26-3120

  Programming Guide, SC26-3113

  Language Reference, SC26-3114

  Reference Summary, SX26-3821

  Diagnosis Guide, SC26-3149

  Compile-Time Messages and Codes, SC26-3229

- **VS COBOL II Publications**

  General Information, GC26-4042

  Licensed Program Specifications, GC26-4044

  Migration Guide for MVS and CMS, GC26-3151

  Installation and Customization for MVS, SC26-4048

  Installation and Customization for CMS, SC26-4213

  Application Program Guide for MVS and CMS, SC26-4045

  Language Reference, GC26-4047

  Reference Summary, SX26-3721

  Debugging, SC26-4049

  Diagnosis Guide, LY27-9523

  Diagnosis Reference, LY27-9522

- **DB2**

  IBM Database 2 Application Programming and SQL Guide, SC26-4889

- **CCCA**

  COBOL and CICS/VS Command Level Conversion Aid, SB11-6432

- **CICS**

  CICS/ESA Installation Guide, SC33-0663

  CICS/ESA Operations Guide, SC33-0668

  CICS/ESA Problem Determination Guide, SC33-0678

  CICS/ESA Definition Manual, SC33-0667

  CICS/ESA Application Programming Guide, SC33-0675

  CICS-IMS Database Control Guide for CICS/ESA, SC33-0660

- **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/SF**

  COBOL Structuring Facility Reference, SC26-3411

- **IMS**

  IMS Application Programming: DL/I Calls, SC26-3062

  Programming language considerations in an IMS Environment—VS COBOL II, G320-9538

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 is composed of 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 Language Environment, 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.

Extension: 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

big-endian. Default format used by the mainframe and the AIX workstation to store binary data. In this format, the least significant digit is on the highest address. Compare with “little-endian.”

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 is normally composed of 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.

Btrieve. A key-indexed record management system that allows applications to manage records by key value, sequential access method, or random access method. IBM COBOL supports COBOL sequential and indexed file I-O language through Btrieve.

buffer. A portion of storage used to hold input or output data temporarily.

built-in function. See “intrinsic function.”

byte. A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character.

C

callable services. In Language Environment, a set of services that can be invoked by a COBOL program using the conventional Language Environment-defined call interface, and usable by all programs sharing the Language Environment 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.

cataloged procedure. A set of job control statements placed in a partitioned data set called the procedure library (SYS1.PROCLIB). You can use cataloged procedures to save time and reduce errors coding JCL.

century window. The 100-year interval in which Language Environment assumes all 2-digit years lie. The Language Environment default century window begins 80 years before the system date.

*** 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**. The entity that defines common behavior and implementation for zero, one, or more objects. The objects that share the same implementation are considered to be objects of the same class.

*** 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 Definition**. The COBOL source unit that defines a class.

*** class identification entry**. An entry in the CLASS-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the class-name and assign selected attributes to the class definition.

*** 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.

class object. The run-time object representing a SOM class.

*** 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
b	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.”

code page. An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for 8-bit code, assignment of characters and meanings to 128 code points for 7-bit code.

*** 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.

compiler directing statement. A statement that specifies actions to be taken by the compiler during processing of a COBOL source program. Compiler directives are contained in the COBOL source program. Thus, you can specify different suboptions of the directive within the source program by using multiple compiler directive statements in the program.

*** complex condition**. A condition in which one or more logical operators act upon one or more conditions. (See also "negated simple condition," "combined condition," and "negated combined condition.")

*** computer-name**. A system-name that identifies the computer upon which the program is to be compiled or run.

condition. An exception that has been enabled, or recognized, by Language Environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and results in an interrupt. They can also be detected by language-specific generated code or language library code.

*** 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 and class definitions.

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.

copybook. A file or library member containing a sequence of code that is included in the source program at compile time using the COPY statement. The file can be created by the user, supplied by COBOL, or supplied by another product.

CORBA. The Common Object Request Broker Architecture established by the Object Management Group. IBM's *Interface Definition Language* used to describe the *interface* for SOM classes is fully compliant with CORBA standards.

*** 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 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 is composed of 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, class definition, or method definition. The DATA DIVISION describes the data to be processed by the object program, class, or method: 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. (Note, the Class DATA DIVISION contains only the WORKING-STORAGE SECTION.)

*** 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 key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of 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 are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

DLL. See "dynamic link library."

do construction. In structured programming, a DO statement is used to group a number of statements in a procedure. In COBOL, an in-line PERFORM statement functions in the same way.

do-until. In structured programming, a do-until loop will be executed at least once, and until a given condition is true. In COBOL, a TEST AFTER phrase used with the PERFORM statement functions in the same way.

do-while. In structured programming, a do-while loop will be executed if, and while, a given condition is true. In COBOL, a TEST BEFORE phrase used with the PERFORM statement functions in the same way.

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. Because each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that 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 link library. A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

Dynamic Storage Area (DSA). Dynamically acquired storage composed of 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 Language Environment.

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 zeroes and/or inserting editing characters.

*** editing character**. A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
b	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. When running under the Language Environment product, an enclave is analogous to a run unit. An enclave can create other enclaves on OS/390 and MVS and CMS by a LINK, on CMS by CMSCALL, and the use of the system() function of C.

***end class header**. A combination of words, followed by a separator period, that indicates the end of a COBOL class definition. The end class header is:

```
END CLASS class-name.
```

***end method header**. A combination of words, followed by a separator period, that indicates the end of a COBOL method definition. The end method header is:

```
END METHOD method-name.
```

*** 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, class definition, or method definition. 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. 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.

environment variable. Any of a number of variables that describe the way an operating system is going to run and the devices it is going to recognize.

execution time. See "run time."

execution-time environment. See "run-time environment."

*** explicit scope terminator**. A reserved word that 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.

extensions. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

*** 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 floating-point item. A format for representing numbers in which a real number is represented by a pair of distinct numerals. In a floating-point representation, the real number is the product of the fixed-point part (the first numeral), and a value obtained by raising the implicit floating-point base to a power denoted by the exponent (the second numeral).

For example, a floating-point representation of the number 0.0001234 is: 0.1234 -3, where 0.1234 is the mantissa and -3 is the exponent.

external program. The outermost program. A program that is not nested.

*** 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 is composed of 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.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

*** 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 is composed of subordinate data items.

H

header label. (1) A file label or data set 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, class definition, or method definition. The IDENTIFICATION DIVISION identifies the program name, class name, or method name. 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 that 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.

IGZCBSN. The COBOL/370 Release 1 bootstrap routine. It must be link-edited with any module that contains a COBOL/370 Release 1 program.

IGZCBSO. The COBOL for MVS & VM Release 2 and COBOL for OS/390 & VM bootstrap routine. It must be link-edited with any module that contains a COBOL for MVS & VM Release 2 or COBOL for OS/390 & VM program.

*** 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 is composed of 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.

*** inheritance (for classes)**. A mechanism for using the implementation of one or more *classes* as the basis for another class. A *subclass* inherits from one or more *superclasses*. By definition the inheriting class conforms to the inherited classes.

*** 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.

inline. In a program, instructions that are executed sequentially, without branching to routines, subroutines, or other programs.

*** 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 or method and that provides information required for transmission and handling of data during execution of the object program or method definition.

*** 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, DISPLAY, OPEN, READ, REWRITE, 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.

instance data. Data defining the state of an object. The instance data introduced by a class is defined in the WORKING-STORAGE SECTION of the DATA DIVISION of the class definition. The state of an object also includes the state of the instance variables introduced by base classes that are inherited by the current class. A separate copy of the instance data is created for each object instance.

*** 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.

interface. The information that a *client* must know to use a *class*—the names of its *attributes* and the signatures of its *methods*. With direct-to-SOM compilers such as COBOL, the interface to a class may be defined by native language syntax for class definitions. Classes implemented in other languages might have their interfaces defined directly in SOM Interface Definition Language (IDL). The COBOL compiler has a compiler option, IDLGEN, to automatically generate IDL for a COBOL class.

Interface Definition Language (IDL). The formal language (independent of any programming language) by which the *interface* for a class of *objects* is defined in a IDL file, which the SOM compiler then interprets to create an implementation template file and binding files. SOM's Interface Definition Language is fully compliant with standards established by the Object Management Group's Common Object Request Broker Architecture (*CORBA*).

interlanguage communication (ILC). The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages.

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.

intrinsic function. A predefined function, such as a commonly used arithmetic function, called by a built-in function reference.

*** 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 that 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.

*** key word**. 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.

Language Environment-conforming. A characteristic of compiler products (COBOL for OS/390 & VM, COBOL for MVS & VM, COBOL/370, AD/Cycle C/370, C/C++ for MVS and VM, PL/I for MVS and VM) that produce object code conforming to the Language Environment format.

last-used state. A program is in last-used state if its internal values remain the same as when the program was exited (are not reset to their initial values).

*** 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 hierarchical 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.

LILIAN DATE. The number of days since the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582. The Lilian date format is named in honor of Luigi Lilio, the creator of the Gregorian calendar.

*** 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.

local. A set of attributes for a program execution environment indicating culturally sensitive considerations, such as: character code page, collating sequence, date/time format, monetary value representation, numeric value representation, or language.

*** LOCAL-STORAGE SECTION**. The section of the DATA DIVISION that defines storage that is allocated and freed on a per-invocation basis, depending on the value assigned in their VALUE clauses.

*** 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.

metaclass. A SOM class whose instances are SOM class-objects. The methods defined in metaclasses are executed without requiring any object instances of the class to exist, and are frequently used to create instances of the class.

method. Procedural code that defines one of the operations supported by an object, and that is executed by an INVOKE statement on that object.

*** Method Definition**. The COBOL source unit that defines a method.

*** method identification entry**. An entry in the METHOD-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the method-name and assign selected attributes to the method definition.

*** method-name**. A user-defined word that identifies a method.

*** 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. When running under the Language Environment product, multitasking is synonymous with *multithreading*.

N

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

O

object. An entity that has state (its data values) and operations (its methods). An object is a way to encapsulate state and behavior.

object code. Output from a compiler or assembler that 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 that 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.

ODBC. Open Database Connectivity that provides you access to data from a variety of databases and file systems.

ODO object. In the example below,

```
WORKING-STORAGE SECTION
01  TABLE-1.
    05  X                   PICS9.
    05  Y OCCURS 3 TIMES
          DEPENDING ON X    PIC X.
```

`X` is the object of the OCCURS DEPENDING ON clause (ODO object). The value of the ODO object determines how many of the ODO subject appear in the table.

ODO subject. In the example above, `Y` is the subject of the OCCURS DEPENDING ON clause (ODO subject). The number of `Y` ODO subjects that appear in the table depends on the value of `X`.

*** 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. (Program IDENTIFICATION DIVISION)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

The permissible paragraph headers in the ENVIRONMENT DIVISION are:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class CONFIGURATION SECTION)
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.

preloaded. In COBOL this refers to COBOL programs that remain resident in storage under IMS instead of being loaded each time they are called.

*** 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, class definition, or method definition. The Procedure Division contains instructions for solving a problem. The Program and Method Procedure Divisions may contain imperative statements, conditional statements, compiler directing statements, paragraphs, procedures, and sections. The Class Procedure Division contains only method definitions.

procedure integration. One of the functions of the COBOL optimizer is to simplify calls to performed procedures or contained programs.

PERFORM procedure integration is the process whereby a PERFORM statement is replaced by its performed procedures. Contained program procedure integration is the process where a CALL to a contained program is replaced by the program code.

*** 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
b	space
=	equal sign

Q

QSAM (Queued 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.

*** qualified data-name**. An identifier that is composed of 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.

*** 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.

recursion. A program calling itself or being directly or indirectly called by a one of its called programs.

recursively capable. A program is recursively capable (can be called recursively) if the RECURSIVE attribute is on the PROGRAM-ID statement.

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 load module.

*** 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.

reusable environment. A reusable environment is when you establish an assembler program as the main program by using either ILBOSTP0 programs, IGZERRE programs, or the RTEREUS run-time option.

routine. A set of statements in a COBOL program that causes the computer to perform an operation or series of related operations. In Language Environment, 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**. A stand-alone object program, or several object programs, that interact via COBOL CALL statements, which function at run time as an entity.

S

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 is composed of 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.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
```

In the Procedure Division, a section header is composed of 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.

SOM. See “System Object Model”

*** 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 is composed of 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 that 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.

STL. STL File System: native workstation and PC file system for COBOL and PL/I. Supports sequential, relative, and indexed files, including the full ANSI 85 COBOL standard I/O language and all of the extensions described in *COBOL Language Reference*, unless exceptions are explicitly noted.

structured programming. A technique for organizing and coding a computer program in which the program comprises 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.

*** subclass**. A class that inherits from another class. When two classes in an inheritance relationship are considered together, the subclass is the inheritor or inheriting class; the *superclass* is the inheritee or inherited class.

*** 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 is composed of a data-name followed by one or more subscripts enclosed in parentheses.

*** superclass**. A class that is inherited by another class. See also *subclass*.

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.

System Object Model (SOM). IBM's object-oriented programming technology for building, packaging, and manipulating class libraries. SOM conforms to the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standards.

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 that are 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 or data set label 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.

universal object reference. A data-name that can refer to an object of any class.

*** 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.

*** variably located group.**. A group item following, and not subordinate to, a variable-length table in the same level-01 record.

*** variably located item.**. A data item following, and not subordinate to, a variable-length table in the same level-01 record.

*** verb**. A word that expresses an action to be taken by a COBOL compiler or object program.

VM/SP (Virtual Machine/System Product). An IBM-licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a "real" machine.

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.

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, composed either of noncontiguous items or working storage records or of both.

Z

zoned decimal item. See "external decimal item."

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

Z

We'd Like to Hear from You

COBOL for OS/390 & VM
COBOL for MVS & VM
Compiler and Run-Time Migration Guide

Publication No. GC26-4764-05

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773.
- Electronic mail—Use one of the following network IDs:
  - IBMLink: COBPUBS at STLVM27
  - Internet: COBPUBS@VNET.IBM.COM

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

COBOL for OS/390 & VM
COBOL for MVS & VM
Compiler and Run-Time Migration Guide

Publication No. GC26-4764-05

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

May we contact you to discuss your comments? Yes No

Would you like to receive our response by E-Mail?

_____

Your E-mail address

Name

Address

Company or Organization

Phone No.

Readers' Comments
GC26-4764-05

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department HHX/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Program Number: 5648-A25

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

COBOL for MVS & VM

SC26-4769	Language Reference
SC26-4767	Programming Guide
GC26-4764	Compiler and Run-Time Migration Guide
SC26-3138	Diagnosis Guide
GC26-4766	Installation and Customization under MVS
GC26-4761	Licensed Program Specification
SC09-2137	Debug Tool User's Guide and Reference
GC26-8664	Fact Sheet

COBOL for OS/390 & VM

SC26-9046	Language Reference
SC26-9049	Programming Guide
GC26-4764	Compiler and Run-Time Migration Guide
SC26-9047	Diagnosis Guide
GC26-9045	Customization under OS/390
GC26-9044	Licensed Program Specification
SC09-2137	Debug Tool User's Guide and Reference
GC26-9048	Fact Sheet

GC26-4764-05

Spine information:

COBOL for OS/390 & VM
COBOL for MVS & VM

Compiler and Run-Time Migration Guide