

Enterprise COBOL for z/OS



Migration Guide

Version 5.11

Enterprise COBOL for z/OS



Migration Guide

Version 5.1.1

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 291.

| **Third edition (March 2019)**

| This edition applies to Version 5 Release 1 Modification 1 of IBM Enterprise COBOL for z/OS (program number 5655-W32) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure that you are using the correct edition for the level of the product.

| You can view or download softcopy publications free of charge at www.ibm.com/shop/publications/order/.

© **Copyright IBM Corporation 1991, 2019.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Tables	vii
-------------------------	------------

Preface	ix
--------------------------	-----------

About this information.	ix
Terminology clarification	ix
IBM COBOL compilers by name and version	ix
Acknowledgement	x
Using your information.	x
Summary of changes to this information.	xi
Changes in GC14-7383-02 (March 2019)	xi
Changes in GC14-7383-00 (June 2013)	xii
Changes in GC23-8527-01 (August 2009).	xii
Changes in GC23-8527-00 (December 2007)	xiii
Changes in GC27-1409-05 (November 2006)	xiii
Changes in GC27-1409-04 (March 2006).	xiii
Changes in GC27-1409-03 (July 2005)	xiii
Changes in GC27-1409-02 (December 2003)	xiv
Changes in GC27-1409-01 (September 2002)	xiv
Changes in GC27-1409-00 (November 2001)	xiv
Changes in GC26-4764-05 (September 2000)	xiv
Summary of changes to the COBOL compilers	xv
Changes in IBM Enterprise COBOL for z/OS, Version 5 Release 1 Modification 1	xv
Changes in IBM Enterprise COBOL for z/OS, Version 5 Release 1	xv
Changes in IBM Enterprise COBOL for z/OS, Version 4 Release 2	xx
Changes in IBM Enterprise COBOL for z/OS, Version 4 Release 1	xx
Changes in IBM Enterprise COBOL for z/OS, Version 3 Release 4: service updates, November 2006	xxi
Changes in IBM Enterprise COBOL for z/OS, Version 3 Release 4	xxii
Changes in IBM Enterprise COBOL for z/OS, Version 3 Release 3	xxiii
Changes in IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 2.	xxiii
Changes in IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 1.	xxiv
Changes in COBOL for OS/390 & VM, Version 2 Release 2	xxv
Changes in COBOL for OS/390 & VM V2 R1 Modification 2.	xxvi
Changes in COBOL for OS/390 & VM V2 R1 Modification 1.	xxvi
Changes in COBOL for OS/390 & VM, Version 2 Release 1	xxvi
How to send your comments.	xxvi
Accessibility	xxvii
Interface information	xxvii
Keyboard navigation	xxvii
Accessibility of this information.	xxvii
IBM and accessibility	xxvii

Part 1. Overview	1
-----------------------------------	----------

Chapter 1. Introducing the new compiler and run time.	3
--	----------

Product relationships: compiler, runtime library, debug	4
Comparison of COBOL compilers	5
Language Environment's runtime support for different compilers	6
Advantages of the new compiler and run time	6
Changes with the new compiler and run time	12
CMPR2 compiler option not available	12
FLAGMIG compiler option	12
SOM-based object-oriented COBOL not available	12
Integrated DB2 coprocessor available	12
Integrated CICS translator available	13
General migration tasks	13
Planning your strategy	13
Upgrading your source to Enterprise COBOL	13
Adding Enterprise COBOL programs to existing applications	15

Chapter 2. Do I need to recompile?	17
---	-----------

Migration basics	17
Runtime migration	17
Source migration	18
Service support for OS/VS COBOL and VS COBOL II programs	18
Changing OS/VS COBOL programs	18
Interoperability with older levels of IBM COBOL programs	19

Part 2. Migration strategies.	21
--	-----------

Chapter 3. Compiler upgrade checklist	23
--	-----------

Chapter 4. Planning to upgrade source programs	25
---	-----------

Preparing to upgrade your source	25
Installing Enterprise COBOL	25
Assessing storage requirements.	25
Deciding which conversion tools to use and install them	26
Educating your programmers on new compiler features.	26
Taking an inventory of your applications	27
Taking an inventory of vendor tools, packages, and products	27
Taking an inventory of COBOL applications	27
Prioritizing your applications	28
Assigning complexity ratings	28
Determining conversion priority	30
Setting up a conversion procedure.	31

Programs without CICS or Report Writer	32
Programs with CICS	33
Programs with Report Writer statements to be discarded	35
Programs with Report Writer statements to be retained	36
Making application program updates.	37

Part 3. Upgrading programs 41

Chapter 5. Upgrading OS/VS COBOL source programs 43

Comparing OS/VS COBOL to Enterprise COBOL	43
Language elements that require change (quick reference)	43
Converting to COBOL 85 Standard	54
COBOL Conversion Tool (CCCA)	54
OS/VS COBOL MIGR compiler option	54
Language elements that require other products for support.	55
Report Writer.	55
Language elements that are not implemented	56
ISAM file handling	56
BDAM file handling	57
Communication feature	57
Language elements that are not supported	58
SEARCH ALL statements.	64
Undocumented OS/VS COBOL extensions that are not supported	64
Language elements that changed from OS/VS COBOL.	72

Chapter 6. Compiling converted OS/VS COBOL programs 89

Compiler options for converted programs	89
Unsupported OS/VS COBOL compiler options	90
Prolog format changes.	91

Chapter 7. Upgrading VS COBOL II source programs 93

Upgrading VS COBOL II programs compiled with the CMPR2 compiler option	93
COBOL 85 Standard interpretation changes.	93
REPLACE and comment lines	94
Precedence of USE procedures	94
Reference modification of a variable-length group receiver.	94
ACCEPT statement.	95
New reserved words	96
New reserved words	96
Undocumented VS COBOL II extensions	97
SEARCH ALL statements.	97
Upgrading programs that use SIMVRD support	97

Chapter 8. Compiling VS COBOL II programs 99

Compiler options for VS COBOL II programs	99
Compiling with Enterprise COBOL	99

Compiler options not supported in Enterprise COBOL.	99
Prolog format changes	100

Chapter 9. Upgrading IBM COBOL source programs 101

Determining which programs require upgrade before you compile with Enterprise COBOL	101
Upgrading programs that have SEARCH ALL statements	102
Upgrading programs that use SIMVRD support	104
Language Environment runtime considerations	105
Numeric items with PICTURE P considerations	105
New reserved words in Enterprise COBOL	105
New reserved words	106
SEARCH ALL statements	106
Migrating from the CMPR2 compiler option to NOCMPR2	107
Upgrading programs compiled with the CMPR2 compiler option	107
Upgrading SOM-based object-oriented (OO) COBOL programs	140
SOM-based OO COBOL language elements that are not supported	140
SOM-based OO COBOL language elements that are changed	141

Chapter 10. Compiling IBM COBOL programs 143

Default compiler options for IBM COBOL programs.	143
Compiler options for IBM COBOL programs	143
Compiler options not available in Enterprise COBOL	145

Chapter 11. Upgrading programs from Enterprise COBOL Version 3. 147

SEARCH ALL statements	147
Upgrading programs that have SEARCH ALL statements	147
Upgrading Enterprise COBOL Version 3 programs that have XML PARSE statements	150
Migrating from the old XML parser to the new XML parser	150
Upgrading Enterprise COBOL programs that have XML GENERATE statements	157
Converting programs that use new reserved words	158
Upgrading programs that use SIMVRD support	158

Chapter 12. Compiling Enterprise COBOL Version 3 programs 161

Compiler option changes from IBM Enterprise COBOL for z/OS, Version 3	161
Differences in the TEST compiler option	161
Debug information changes with IBM Enterprise COBOL Version 5	162

Chapter 13. Upgrading from Enterprise COBOL Version 4. 165

Upgrading Enterprise COBOL Version 4 programs that have XML PARSE statements	165
Migrating from the old XML parser to the new XML parser	166
Upgrading Enterprise COBOL Version 4 Release 1 programs that have XML PARSE statements and that use the XMLPARSE(XMLSS) compiler option	173
Changes in millenium language extensions in IBM Enterprise COBOL for z/OS, Version 5	174

Chapter 14. Compiling Enterprise COBOL V4 programs 175

Compiler option changes from IBM Enterprise COBOL for z/OS, Version 4	175
Debug information changes with IBM Enterprise COBOL Version 5	175

Part 4. What is new and different with Enterprise COBOL V5.1? . . . 177

Chapter 15. Changes with IBM Enterprise COBOL for z/OS, Version 5.1 179

Prerequisite software and service for Enterprise COBOL V5	179
COBOL source code differences in Enterprise COBOL V5.1	180
Compiler option changes in IBM Enterprise COBOL for z/OS, Version 5	181
Changes in compiling with Enterprise COBOL Version 5.1	182
Compiler output to uninitialized data sets not supported	184
JCL and packaging changes for Enterprise COBOL V5.1	185
Link edit/bind changes with Enterprise COBOL Version 5.1	185
Changes at run time with Enterprise COBOL V5.1	186
Language Environment option changes.	188
Variable length records - wrong length READ	188
Interoperability with older levels of IBM COBOL programs	190
Debug information changes with IBM Enterprise COBOL Version 5	191
WORKING-STORAGE SECTION changes	192

Chapter 16. Adding Enterprise COBOL V5.1 programs to existing COBOL applications 197

AMODE and RMODE considerations	199
--	-----

Part 5. Enterprise COBOL migration and other IBM products . 201

Chapter 17. Debug tool 203

Initiating Debug Tool.	203
Debug information changes with IBM Enterprise COBOL Version 5	204
Debug Tool changes with IBM Enterprise COBOL Version 5	204
Full Screen Mode changes with IBM Enterprise COBOL V5.1	208
Debug Tool changes for remote mode with IBM Enterprise COBOL V5.1	209

Chapter 18. CICS conversion considerations for COBOL source . . 211

CSD setup differences with Enterprise COBOL V5	211
DFHRPL setup differences with Enterprise COBOL V5	212
Compiler options for programs that run under CICS	213
Migrating from the separate CICS translator to the integrated translator	214
Integrated CICS translator	214

Chapter 19. DB2 coprocessor conversion considerations 217

DB2 coprocessor integration	217
Language elements	219
Code-page conversion	222

Chapter 20. Moving IMS programs to Enterprise COBOL V5. 223

Compiling and linking for running under IMS	223
---	-----

Part 6. Appendixes 225

Appendix A. Commonly asked questions and answers 227

Compatibility	227
Compiling with Enterprise COBOL	228
Binding (link-editing) Enterprise COBOL programs	229
Language Environment services	229
Language Environment runtime options	229
Subsystems	230
z/OS	231
Performance.	232
Service	232
Object-oriented syntax, and Java 5, Java 6 and Java 7 SDKs	232

Appendix B. COBOL reserved word comparison 233

Appendix C. Conversion tools for source programs 249

MIGR compiler option	249
Language differences	249
Statements supported with enhanced accuracy	250
LANGLVL(1) statements not supported	251

LANGLVL(1) and LANTLRVL(2) statements not supported	251
Other programs that aid conversion	253
Rational Asset Analyzer	253
COBOL and CICS/VS Command Level Conversion Aid (CCCA).	253
COBOL Report Writer Precompiler	255
Debug Tool Load Module Analyzer	256
The Edge Portfolio Analyzer	256

Appendix D. Applications with COBOL and assembler 257

Called assembler programs	257
SVC LINK and COBOL run-unit boundary	257
Runtime support for assembler COBOL calls under non-CICS.	258
Runtime support for assembler COBOL calls under CICS	259
Converting programs that change the program mask	260
Upgrading applications that use an assembler driver	260
Convert the assembler driver	261
Modify the assembler driver	261
Use an unmodified assembler driver	261
Assembler programs that load and BALR to MAIN COBOL programs	261
Assembler programs that load and delete COBOL programs.	261

Appendix E. Option comparison . . . 263

Appendix F. Compiler limit comparison 279

Appendix G. Preventing file status 39 for QSAM files 283

Processing existing files	283
Defining variable-length records	283

Defining fixed-length records	284
Converting existing files that do not match the COBOL record	284
Processing new files	284
Processing files dynamically created by COBOL	285

Appendix H. TSO considerations . . . 287

Using REXX execs.	287
---------------------------	-----

Appendix I. Accessing JCL parameters 289

Notices 291

Programming interface information	293
Trademarks	293

Glossary 295

List of resources 323

IBM Enterprise COBOL for z/OS.	323
Related publications	323

Index 325

Tables

1. COBOL compiler name, version, release and product numbers	ix	26. The predefined entity references	157
2. The Enterprise COBOL for z/OS publications	x	27. Steps for using variable-length RRDS	158
3. The Language Environment element of z/OS publications	xi	28. Compiler options not available in Enterprise COBOL Version 5	161
4. Comparison of COBOL compilers	5	29. The deprecated TEST suboptions	162
5. Advantages of Enterprise COBOL and Language Environment	6	30. The predefined entity references	173
6. Complexity ratings for program attribute conversions	29	31. Compiler options not available in Enterprise COBOL Version 5	175
7. Assigning program conversion priorities	30	32. Compiler options not available in Enterprise COBOL Version 5	181
8. Language element differences between OS/VS COBOL and Enterprise COBOL	44	33. Compiler options new and changed with Enterprise COBOL Version 5	181
9. Rules for VSAM file definitions	61	34. Runtime option changes with Enterprise COBOL Version 5	188
10. Status key values: QSAM files	77	35. Area where WORKING-STORAGE is located	194
11. Status key values: VSAM files	77	36. How to find the PPA4, NORENT static area, LE's WSA, RENT static area, and program static area in a dump?	194
12. USE FOR DEBUGGING declarative: valid operands	84	37. Compiler options for programs that run under CICS	213
13. Compiler options for converted OS/VS COBOL programs	89	38. Key compiler options for the integrated CICS translator	215
14. OS/VS COBOL compiler options not supported by Enterprise COBOL	90	39. Recommended compiler options for applications with mixed COBOL programs.	224
15. New reserved words, by compiler.	96	40. Reserved word comparison	233
16. Steps for using variable-length RRDS	97	41. COBOL statements dealing with primary BLLs	255
17. Key Enterprise COBOL compiler options for VS COBOL II programs	99	42. Language Environment supported calls between COBOL programs and assembler programs under non-CICS; Yes indicates that a call is supported.. . . .	258
18. Compiler options not supported in Enterprise COBOL	100	43. Language Environment supported calls between COBOL programs and assembler programs that run under CICS; Yes indicates that a call is supported.	259
19. Steps for using variable-length RRDS	104	44. Option comparison	263
20. New reserved words, by compiler.	106		
21. Language elements different between CMPR2 and NOCMPR2	108		
22. QSAM and VSAM file status codes with CMPR2 and NOCMPR2	115		
23. Rules for VSAM file definitions	119		
24. Compiler options for IBM COBOL programs	143		
25. Compiler options not available in Enterprise COBOL	145		

Preface

About this information

This information provides topics to help you to move to IBM® Enterprise COBOL Version 5.1.

This information assumes that you have completed your runtime migration to Language Environment®.

Terminology clarification

In this information, we use the term Enterprise COBOL as a general reference to:

- IBM Enterprise COBOL for z/OS® and OS/390®, Version 3 Release 1
- IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
- IBM Enterprise COBOL for z/OS, Version 3 Release 3
- IBM Enterprise COBOL for z/OS, Version 3 Release 4
- IBM Enterprise COBOL for z/OS, Version 4 Release 1
- IBM Enterprise COBOL for z/OS, Version 4 Release 2
- IBM Enterprise COBOL for z/OS, Version 5 Release 1

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

- COBOL/370, Version 1 Release 1
- 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

See “Summary of changes to the COBOL compilers” on page xv for further details.

IBM COBOL compilers by name and version

Table 1. COBOL compiler name, 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
Enterprise COBOL for z/OS	Version 3 Release 1	5655-G53
Enterprise COBOL for z/OS	Version 3 Release 2	5655-G53
Enterprise COBOL for z/OS	Version 3 Release 3	5655-G53
Enterprise COBOL for z/OS	Version 3 Release 4	5655-G53
Enterprise COBOL for z/OS	Version 4 Release 1	5655-S71
Enterprise COBOL for z/OS	Version 4 Release 2	5655-S71
Enterprise COBOL for z/OS	Version 5 Release 1	5655-W32

To aid in moving your runtime library to Language Environment, you can find 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 runtime options for compatible behavior in the *Enterprise COBOL V4.2 Compiler and Runtime Migration Guide* at <http://publibfp.dhe.ibm.com/epubs/pdf/igy3mg50.pdf>.

To aid in moving from your older COBOL compiler to Enterprise COBOL, this information provides descriptions of the language differences between older COBOL compilers and Enterprise COBOL and describes the IBM conversion tools available to aid in converting your source programs to Enterprise COBOL programs. It also describes other differences that might require changes in your application development process in order to use Enterprise COBOL.

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 Migration Guide. Without such assistance, this information would have been much more difficult to develop.

Using your information

The information provided with Enterprise COBOL and Language Environment is designed to help you do COBOL programming under z/OS.

Enterprise COBOL for z/OS Version 5 Release 1

Table 2. The Enterprise COBOL for z/OS publications

Task	Information
Understand warranty information	<i>Licensed Program Specifications</i>
Install the compiler under z/OS	<i>Program Directory for Enterprise COBOL</i>
Understand product changes and upgrade source to the latest version of Enterprise COBOL for z/OS	<i>Migration Guide</i> .
Upgrade run time environment to Language Environment	Note: If you have not yet migrated your runtime library to Language Environment, consult the <i>Enterprise COBOL V4.2 Compiler and Runtime Migration Guide</i> at http://publibfp.dhe.ibm.com/epubs/pdf/igy3mg50.pdf for help.
Customize Enterprise COBOL for z/OS	<i>Enterprise COBOL Customization Guide</i>
Prepare and test your programs and get details about compiler options	<i>Enterprise COBOL Programming Guide</i>
Get details about COBOL syntax and specifications of language elements	<i>Enterprise COBOL Language Reference</i>

Language Environment element of z/OS

Table 3. The Language Environment element of z/OS publications

Task	Information
Evaluate the product	<i>Language Environment Concepts Guide</i>
Install Language Environment	<i>z/OS Program Directory</i>
Understand Language Environment program models and concepts	<i>Language Environment Programming Guide</i>
Find syntax for Language Environment runtime options and callable services	<i>Language Environment Programming Reference</i>
Debug applications that run with Language Environment, get details about runtime messages, and diagnose problems with Language Environment	<i>Language Environment Debugging Guide and Run-Time Messages</i>
Migrate applications from one release of Language Environment to another.	<i>Language Environment Run-Time Migration Guide</i>
Develop interlanguage communication (ILC) applications	<i>Language Environment Writing Interlanguage Applications</i>
Learn about the concepts and use of Common Debug Architecture (CDA)	<i>Common Debug Architecture User's Guide</i>
Get details about APIs in the Debug Data Program Information library (llibddpi).	<i>Common Debug Architecture Library Reference</i>
Get details about the IBM extensions to the DWARF and ELF APIs in the DWARF 4 standard.	<i>DWARF/ELF Extensions Library Reference</i>

Summary of changes to this information

This section lists the major changes that have been made to each edition of this migration guide since IBM COBOL for OS/390 & VM Version 2 Release 1.

Changes in GC14-7383-02 (March 2019)

Changes in GC14-7383-02 (March 2019)

- Added a support page link where lists the Enterprise COBOL V4 PTFs to support the migration to Enterprise COBOL V5 or V6. For details, see Chapter 13, "Upgrading from Enterprise COBOL Version 4," on page 165.

Changes in GC14-7383-02 (September 2018)

- Added information about how to find WORKING-STORAGE SECTION in Enterprise COBOL V5. For details, see "WORKING-STORAGE SECTION changes" on page 192.

Changes in GC14-7383-02 (May 2018)

- Updated information in "CSD setup differences with Enterprise COBOL V5" on page 211 because certain CICS TS versions provide the *system autoinstall* capability for LE programs and CICS will create the program definition automatically when the programs are first loaded.
- Clarified that the separate CICS translator is still shipped with current CICS products but is no longer being enhanced in "Migrating from the separate CICS translator to the integrated translator" on page 214.

Changes in GC14-7383-02 (April 2017)

- Updated information about calling a ILB0ABN0 callable service with Enterprise COBOL V5 and later versions in “Changes at run time with Enterprise COBOL V5.1” on page 186.

Changes in GC14-7383-02 (February 2017)

- Added information about a new warning message that will be issued when a call to ILB0ABN0 callable service is encountered in the source program in “Changes at run time with Enterprise COBOL V5.1” on page 186.

Changes in GC14-7383-02 (March 2014)

Added back the support for AMODE 24 execution of COBOL programs, except for a few exception cases. Many programs that are compiled by Enterprise COBOL 5.1.1 execute in either AMODE 31 or AMODE 24.

Changes in GC14-7383-00 (June 2013)

This migration guide has been reorganized for Enterprise COBOL Version 5.1. If you have not yet completed your runtime migration to Language Environment, please refer to the previous version of this information. You can use the *Enterprise COBOL V4.2 Compiler and Runtime Migration Guide* at <http://publibfp.dhe.ibm.com/epubs/pdf/igy3mg50.pdf> for help in completing your runtime migration.

Primarily, the following changes have been made to this Migration Guide:

- Removal of the information related to Language Environment
- Addition of specific chapters for migrating from Enterprise COBOL Version 3 and Enterprise COBOL Version 4
- Addition of a section on Enterprise COBOL Version 5
- Addition of a section on upgrading your COBOL compiler along with other IBM products. That includes information about Debug Tool, CICS®, and Db2®. Please see Part 5, “Enterprise COBOL migration and other IBM products,” on page 201 for more information.

There is a lot of information in this guide but most of it is not needed by most customers. For example, if you are moving from Enterprise COBOL Version 4 and you have completed your runtime migration for all applications, you only need to look at a few sections. For details, see Chapter 13, “Upgrading from Enterprise COBOL Version 4,” on page 165, Chapter 14, “Compiling Enterprise COBOL V4 programs,” on page 175, and Chapter 15, “Changes with IBM Enterprise COBOL for z/OS, Version 5.1,” on page 179.

Changes in GC23-8527-01 (August 2009)

Compiler

- Added information about integrated Db2 coprocessor
- Updated information about migrating from XMLPARSE(COMPAT) to XMLPARSE(XMLSS), for example, changes in the handling of several XML events
- Updated information about the differences in parsing behavior when you compile using XMLPARSE(XMLSS)
- Added new reserved words
- Added new compiler options

- Added information to appendix on commonly asked questions and answers:
 - Information about COBOL program calls
 - Information about running existing COBOL applications with Java™ 5 or Java 6

Run time

- Updated information about region-wide defaults
- Updated information about TEST option
- Updated information about Language Environment STORAGE(00) option

Information about CICS has been corrected.

Miscellaneous maintenance and editorial changes have been made; for example, Appendix B, “COBOL reserved word comparison,” on page 233 and Appendix F, “Compiler limit comparison,” on page 279 have been updated.

Changes in GC23-8527-00 (December 2007)

Compiler

- Added section on migrating XML PARSE from Enterprise COBOL Version 3 to Enterprise COBOL Version 4 (Migrating from XMLPARSE(COMPAT) to XMLPARSE(XMLSS)).
- Added information about the new TEST suboptions of Enterprise COBOL Version 4
- Added new reserved words
- Added information to section on migrating from CMPR2 to NOCMPR2:
 - Fixed file attributes and DCB= parameters of JCL
 - OPEN statement failing for QSAM files (FILE STATUS 39)
 - OPEN statement failing for VSAM files (FILE STATUS 39)
- Added information to appendix on DB2® coprocessor integration
 - Additional differences from separate precompiler

Run time

- Added information about removal of SIMVRD runtime option support for Enterprise COBOL Version 4 Release 1 programs.

Changes in GC27-1409-05 (November 2006)

- Updated the documentation of differences between Db2 precompiler and coprocessor.
- Added compiler option SQLCCSID.

Changes in GC27-1409-04 (March 2006)

- Added to the documentation of differences between Db2 precompiler and coprocessor.
- Added a section on migrating SEARCH ALL statements to V3R4.

Changes in GC27-1409-03 (July 2005)

- Added compiler option MDECK.
- Added new reserved words.
- Added SQL code differences between Db2 precompiler and coprocessor.

- Changes to data-item sizes.

Changes in GC27-1409-02 (December 2003)

- Applied service updates to the information

Changes in GC27-1409-01 (September 2002)

Compiler

- Added information about the use of the SEPARATE suboption with the TEST(. . .,SYM,. . .) compiler option.

Run time

- Clarified the information about file handling for COBOL programs with RECORDING MODE U under OS/390, Version 2 Release 10.
- Added information about the change in the amount of space that is used for an output file that is defined as RECFM=U under OS/390, Version 2 Release 10.
- Added information about dynamic calls to assembler programs under Language Environment for z/OS, Version 1 Release 2 and later.

Changes in GC27-1409-00 (November 2001)

Compiler

- Removed various compiler options, including the CMPR2 compiler option
- Added new reserved words
- Added information about the new integrated CICS translator
- Removed OO COBOL syntax and programming model based on SOM
- Added information about migrating to the Enterprise COBOL compiler

Run time

- Added information about the change in behavior for DATA(31) programs
- Added information about CEEDUMP absent from applications with assembler programs that use the DUMP macro
- Added information about the change in file handling for COBOL programs with RECORDING MODE U
- Added information about calling between assembler and COBOL

Changes in GC26-4764-05 (September 2000)

Compiler

- Added newly discovered undocumented extensions and improved many existing entries in Chapter 5, "Upgrading OS/VS COBOL source programs," on page 43
- Added new reserved words
- Added information about migrating to the V2R2 compiler

Run time

- Added description of the new default for runtime 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 runtime options

- Updated the virtual storage requirements
- Updated the CICS considerations:
 - Performance
 - SORT interface change
 - DISPLAY statement
- Updated information about upgrading Language Environment release levels

Miscellaneous maintenance and editorial changes have been made.

Summary of changes to the COBOL compilers

This section lists the main changes that have been made to IBM host COBOL compilers.

Changes in IBM Enterprise COBOL for z/OS, Version 5 Release 1 Modification 1

- Except for a few exception cases, AMODE 24 execution of COBOL programs is supported. Many programs compiled by IBM Enterprise COBOL for z/OS V5.1.1 will execute in AMODE 31 or AMODE 24.
- A new compiler option, SQLIMS, enables the new IMS SQL coprocessor (called SQL statement coprocessor by IMS). The new coprocessor handles your source programs that contain embedded SQLIMS statements.
- New fatal and warning exception codes are added for XML PARSE exceptions.
- The LIST option output in the compiler listing contains a new Special Register Table that provides the location information for all the COBOL Special Register variables.

Changes in IBM Enterprise COBOL for z/OS, Version 5 Release 1

New and changed COBOL function

The XML function supported by IBM Enterprise COBOL for z/OS has been enhanced:

- The XML GENERATE statement has been extended with new syntax that gives the programmer more flexibility and control over the form of the XML document that is generated:
 - The NAME phrase has been added to allow user-supplied element and attribute names.
 - The TYPE phrase has been added to give the user control of attribute and element generation.
 - The SUPPRESS phrase has been added to allow suppression of empty attributes and elements.
- XML parsing support has been enhanced with a special register, XML-INFORMATION, to easily determine whether the XML content delivered for an XML event is complete or will be continued on the next event.
- The compatibility-mode COBOL XML parser from the COBOL library is no longer supported for use by Enterprise COBOL V5 programs. XML PARSE statements in V5 programs always use the XML parser in z/OS XML System Services.

New support for UNBOUNDED tables and groups enables top-down mapping of data structures between XML and COBOL applications

Unicode support has been enhanced in this release with the addition of 6 new intrinsic function:

- ULENGTH
- UPOS
- USUBSTR
- USUPPLEMENTARY
- UVALID
- UWIDTH

A new inline comment indicator (the character string '*>') can be coded to indicate that the ensuing text on a line is a comment.

Enterprise COBOL Version 5.1 corrects READ statement processing of wrong-length records.

The Millennium Language Extensions are no longer supported, and the removed elements are:

- DATEVAL intrinsic function
- UNDATE intrinsic function
- YEARWINDOW intrinsic function
- DATEPROC compiler option
- YEARWINDOW compiler option

To be compatible with the convention used by C and C++, the linkage convention for returning a doubleword binary item specified in the RETURNING phrase PROCEDURE DIVISION header and the CALL statement is changed.

Format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and the syntax: GO TO MORE-LABELS are no longer supported.

Option changes

- The following compiler options are new:
 - AFP(VOLATILE | NOVOLATILE)
 - ARCH(*n*)
 - DISPSIGN(SEP | COMPAT)
 - HGPR(PRESERVE | NOPRESERVE)
 - MAXPCF(*nnn*)
 - STGOPT | NOSTGOPT
- The following compiler options are modified:
 - The MDECK option no longer has a dependency on the LIB option, as the compiler behaves as though the LIB option is always enabled.
 - The MIG suboption of the NUMPROC compiler option is no longer supported
 - The compiled-in range checks cannot be disabled at run time using the runtime option CHECK(OFF).
 - Execution of NORENT programs above the 16 MB line is not supported.

- The HOOK | NOHOOK and SEPARATE | NOSEPARATE suboptions of the TEST compiler option are no longer supported. Those suboptions continue to be tolerated to ease migration. New suboptions SOURCE and NOSOURCE are added to the TEST compiler option.
- The NOSTEST option is enhanced to include the suboptions DWARF and NODWARF.
- The EXIT compiler option is no longer mutually exclusive with the DUMP compiler option, and the compiler exits rules are updated.
- The OPTIMIZE option is modified to allow several level of optimization. The previous OPTIMIZE option format is deprecated but is tolerated for compatibility.
- The format and contents of listing generated from the LIST option are new
- The format and contents of the listing output generated from the MAP option are changed
- Support for the following compiler options has been removed:
 - DATEPROC
 - LIB
 - SIZE(MAX)
 - YEARWINDOW
 - XMLPARSE

Compiler behavior changes

There have been a number of changes to Enterprise COBOL V5.1 that result in different behaviors.

- AMODE 24 execution of programs compiled with Enterprise COBOL V5.1.0 is no longer supported. Enterprise COBOL V5.1.0 executable modules must be AMODE 31.
- The IGZERRE and ILB0STP0 interfaces for managing a reusable COBOL environment are not supported for applications containing programs compiled with Enterprise COBOL V5.
- The IGZBRDGE macro, for converting static calls to dynamic calls, is not supported for programs compiled with Enterprise COBOL V5.
- The compatibility-mode COBOL XML parser from the COBOL library, the old parser from Enterprise COBOL V3, is no longer supported for use by Enterprise COBOL V5 programs. XML PARSE statements in V5 programs always use the z/OS System Services XML parser (XMLSS).
- Enterprise COBOL Version 5 now requires Language Environment at compilation time. If the Language Environment data sets SCEERUN and SCEERUN2 are not installed in the MVS LNKST or LPALST, they must be included in the STEPLIB or JOBLIB concatenation for the compilation.
- Enterprise COBOL Version 5.1 has a new Language Environment member ID, 4. Prior versions of COBOL use ID 5.
- Enterprise COBOL Version 5 programs have some restrictions with interoperability with older versions of COBOL. For details see, “Interoperability with older levels of IBM COBOL programs” on page 19.
- COBOL programs with the following characteristics may behave differently with Enterprise COBOL V5 than with prior versions:
 - Programs that use unsupported COBOL language syntax.
 - Programs referencing data items that, at run time, contain values not conforming to the PICTURE clause on the data description entry. For example:

- a fullword binary item with picture S9(6) USAGE BINARY, containing an oversize value of +123456789 (unless the TRUNC(BIN) option was specified)
 - a two-byte PACKED-DECIMAL item with picture S99, containing an oversize value of 123 (such as, 123C in hexadecimal).
 - a packed decimal or zoned decimal item containing an invalid or non-preferred sign, that does not conform to the sign requirements of the data description entry and the NUMPROC(PFD) compiler option setting in effect.
 - Programs with undiagnosed subscript range errors (when the SSRANGE compiler option was not specified), that reference storage outside the storage allocation for the base data item.
 - Applications with low-level dependencies on specific generated code sequences, register conventions, or internal IBM control blocks may behave differently with Enterprise COBOL V5 than with prior versions.
 - It is illegal to specify a value greater than integer-2 for the object of an OCCURS DEPENDING ON clause, and thus the behavior is undefined. However, Enterprise COBOL V5.1 behaves differently than prior versions when it occurs.
- VSAM record areas for reentrant COBOL programs are allocated above 16 MB if DATA(31) is enabled. Programs that pass data in VSAM file records as CALL ... USING BY REFERENCE parameters to AMODE 24 subprograms may be impacted. Such programs can be recompiled with the DATA(24) compiler option, or the Language Environment HEAP(BELOW) option can be used, to ensure that the records are addressable by the AMODE 24 programs.
 - Compile-time storage requirements are substantially increased, compared to prior versions of Enterprise COBOL. See the discussion of the SIZE option. This is particularly true at higher optimization levels, that is, programs compiled with the OPT(1) or OPT(2) compiler option.
 - Compile-time CPU time requirements are substantially increased, compared to prior versions of Enterprise COBOL.
 - Compile time and run time diagnostic messages may differ, and may be generated at different times or locations.
 - Presence or absence of informational and warning level diagnostics may differ
 - Diagnostics for programs that define excessive and unsupported amounts of storage may be diagnosed either by the binder at bind time, or by Language Environment at run time, instead of by the compiler at compilation time.
 - Compiler listing format and contents differ from prior versions of Enterprise COBOL.

Application performance changes

The OPTIMIZE option has been changed to support several levels of performance optimization for your application. The suboptions have also been changed. The previous OPTIMIZE option format is deprecated but is tolerated for compatibility.

Note: Although OPT(0) is equivalent to the old NOOPTIMIZE option in most ways, OPT(0) removes some unreachable code that was not previously removed with NOOPTIMIZE.

Debugging changes

When the TEST option is specified, DWARF debugging information is included in the application module.

With NOLOAD debug segments in the program object, Enterprise COBOL V5 debug data always matches the executable file, and is always available without giving lists of data sets to search, and does not increase the size of the loaded program.

If you specify the TEST(SOURCE) option, the DWARF debug information includes the expanded source code, and the compiler listing is not needed by IBM Debug Tool. When the TEST(NOSOURCE) is specified, the generated DWARF debugging information does not include the expanded source code.

You can use the NOTEST(DWARF) option to include basic DWARF diagnostic information in the application module. This enables application failure analysis tools, such as CEEDUMP and IBM Fault Analyzer.

Packaging and JCL changes

There have been a number of changes to the packaging, installation and JCL with Enterprise COBOL V5.1.

The SIGYCOMP data set is now a PDSE, rather than a PDS data set as in prior versions.

Enterprise COBOL Version 5.1 requires additional data sets

- When compiling under z/OS TSO or batch, the COBOL compiler now requires 15 utility data sets, SYSUT1 to SYSUT15
- The SYSMDECK data set is now required for all compilations. SYSMDECK may be specified as a utility (temporary) dataset if the NOMDECK option is specified. When MDECK(...) is specified, the SYSMDECK DD allocation must specify a permanent data set.
- The alternate DDNAME list parameter used when the COBOL compiler is invoked from an assembly language program has been expanded with entries for the additional work data sets.

The catalogued procedures that ship with Enterprise COBOL Version 5.1 have been modified.

- IGYWC
- IGYWCL
- IGYWCLG

The following JCL catalogued procedures are no longer supported. Because they all use the Language Environment Prelinker or the DFSMS Loader, which are no longer supported.

- IGYWCG
- IGYWCPG
- IGYWCPL
- IGYWCPLG
- IGYWPL

Restrictions

If you use COBOL for IMS exit routines, Enterprise COBOL V5.1 can compile programs only when the exit is an assembler program in a PDS data set that

LOADs and calls a COBOL V5.1 program in a PDSE. For workarounds to handle the restriction, see Chapter 20, “Moving IMS programs to Enterprise COBOL V5,” on page 223.

•

Changes in IBM Enterprise COBOL for z/OS, Version 4 Release 2

- New and enhanced XML PARSE capabilities are available when you use the z/OS System Services XML parser:
 - You can parse documents with validation against an XML schema when you use the VALIDATING phrase of the XML PARSE statement.
 - The performance of nonvalidating parsing with the XMLPARSE(XMLSS) compiler option is improved compared to the performance of nonvalidating parsing with the XMLPARSE(XMLSS) compiler option in Enterprise COBOL Version 4 Release 1.
 - Character processing is enhanced for any XML document that contains a reference to a character that is not included in the single-byte EBCDIC code page of the document.
- A facility for customizing compiler messages (changing their severity or suppressing them), including FIPS (FLAGSTD) messages, is made possible by a new suboption, MSGEXIT, of the EXIT compiler option.
- A new compiler option, BLOCK0, activates an implicit BLOCK CONTAINS 0 clause for all eligible QSAM files in your program.
- The underscore character (_) is now supported in user-defined words such as data-names and program-names. Underscores are also supported in the literal form of program-names.
- If you use the integrated CICS translator, the compiler listing will now show the CICS options that are in effect.
- Enterprise COBOL applications that use object-oriented syntax for Java interoperability are now supported with Java 5 and Java 6 in addition to the Java SDK 1.4.2.

Changes in IBM Enterprise COBOL for z/OS, Version 4 Release 1

- The XML GENERATE statement has been extended with new syntax that gives the programmer more flexibility and control over the form of the XML document that is generated:
 - The WITH ATTRIBUTES phrase, which causes eligible items in the XML document to be generated as XML attributes instead of as elements.
 - The WITH ENCODING phrase, which allows the user to specify the encoding of the generated document.
 - The WITH XML-DECLARATION phrase, which causes the version and encoding information to be generated in the document.
 - The NAMESPACE and NAMESPACE-PREFIX phrases, which allow generation of XML documents that use XML namespaces.
 - The XML GENERATE statement now supports generation of XML documents encoded in UTF-8 Unicode.
- XML PARSE support has been enhanced:
 - The z/OS System Services XML parser is now supported as an alternative to the existing XML parser that is part of the COBOL library

- The z/OS System Services XML parser provides the following benefits:
 - Availability of the latest IBM parsing technology for COBOL users.
 - Offloading of COBOL XML parsing to zAAP specialty processors.
 - Improved support for parsing XML documents that use XML namespaces.
 - Direct support for parsing XML documents that are encoded in UTF-8 Unicode.
 - Support for parsing very large XML documents, a buffer at a time.
- Four new special registers are introduced for namespace processing during execution of XML PARSE statements.
- The XML PARSE statement has been extended with new syntax. The new WITH ENCODING and RETURNING NATIONAL phrases give the programmer control over the assumed encoding of input XML documents, to facilitate parsing in Unicode.
- A new compiler option, XMLPARSE, has been created to control whether the z/OS System Services parser or the existing COBOL parser is used for XML PARSE statements. With the XMLPARSE(COMPAT) option, XML parsing is fully compatible with Enterprise COBOL Version 3. With the default XMLPARSE(XMLSS) option, the z/OS System Services parser is used and new XML parsing capabilities are enabled.
- Performance of COBOL application programs has been enhanced by exploitation of new z/Architecture[®] instructions. The performance of COBOL Unicode support (USAGE NATIONAL data) has been significantly improved.
- DB2 support has been enhanced in this release, including DB2 V9 exploitation and improvements in coprocessor integration and usability:
 - Support for new SQL data types and new SQL syntax provided by DB2 V9
 - DB2 precompiler options are shown in the compiler listing (DB2 V9 only)
 - SQLCA and SQLDA control blocks are expanded in the compiler listing (all DB2 releases)
 - A new compiler option SQLCCSID is provided to coordinate the coded character set id (CCSID) between COBOL and DB2
- Support for DFSMS large-format data sets
- Debugging enhancements:
 - Debug Tool V8 enablement, new debugging commands
 - GOTO/JUMPTO in optimized code, new TEST suboption EJPD
- Compiler options can be specified in a data set (OPTFILE option)
- Cross-reference of COPY statements, libraries, and data sets in compiler listing

Changes in IBM Enterprise COBOL for z/OS, Version 3 Release 4: service updates, November 2006

- PK31411: A new compiler option, SQLCCSID, which works in conjunction with the DB2 coprocessor, determines whether the CODEPAGE compiler option influences the processing of SQL statements in COBOL programs. SQLCCSID was added via APAR PK31411.
- PK16765: Corrections to the behavior of the SEARCH ALL statement have been made.

With current service applied, specifically the PTF for APAR PK16765, new compiler diagnostic messages and runtime diagnostic messages have been added to assist in identifying programs and SEARCH ALL statements that are potentially impacted by these corrections and may require modification in order

to migrate to V3R4. If you have this PTF on your compiler, the listing header and object program will show Version 3 Release 4 Modification 1.

Changes in IBM Enterprise COBOL for z/OS, Version 3 Release 4

- Several limits on COBOL data-item size have been significantly raised, for example:
 - The maximum data-item size has been raised from 16 MB to 128 MB.
 - The maximum PICTURE symbol replication has been raised to 134,217,727.
 - The maximum OCCURS integer has been raised to 134,217,727.(For full details about changed compiler limits, see the COBOL Language Reference.) This support facilitates programming with large amounts of data, for example:
 - DB2/COBOL applications that use DB2 BLOB and CLOB data types
 - COBOL XML applications that parse or generate large XML documents
- Support for national (Unicode UTF-16) data has been enhanced. Several additional kinds of data items can now be described implicitly or explicitly as USAGE NATIONAL:
 - External decimal (*national decimal*) items
 - External floating-point (*national floating-point*) items
 - Numeric-edited items
 - National-edited items
 - Group (*national group*) items, supported by the GROUP-USAGE NATIONAL clause
- Many COBOL language elements support the new kinds of UTF-16 data, or newly support the processing of national data:
 - Numeric data with USAGE NATIONAL (national decimal and national floating point) can be used in arithmetic operations and in any language constructs that support numeric operands .
 - Edited data with USAGE NATIONAL is supported in the same language constructs as any existing edited type, including editing and de-editing operations associated with moves.
 - Group items that contain all national data can be defined with the GROUP-USAGE NATIONAL clause, which results in the group behaving as an elementary item in most language constructs. This support facilitates use of national groups in statements such as STRING, UNSTRING, and INSPECT.
 - The XML GENERATE statement supports national groups as receiving data items, and national-edited, numeric-edited of USAGE NATIONAL, national decimal, national floating-point, and national group items as sending data items.
 - The NUMVAL and NUMVAL-C intrinsic functions can take a national literal or national data item as an argument.Using these new national data capabilities, it is now practical to develop COBOL programs that exclusively use Unicode for all application data.
- The REDEFINES clause has been enhanced such that for data items that are not level 01, the subject of the entry can be larger than the data item being redefined.
- A new compiler option, MDECK, causes the output from library-processing statements to be written to a file .
- DB2 coprocessor support has been enhanced: XREF is improved.

- The literal in a VALUE clause for a data item of class national can be alphanumeric .

These terminology changes were also made in this release:

- The term *alphanumeric group* is introduced to refer specifically to groups other than national groups.
- The term *group* means both alphanumeric groups and national groups except when used in a context that obviously refers to only an alphanumeric group or only a national group.
- The term *external decimal* refers to both zoned decimal items and national decimal items.
- The term *alphanumeric floating point* is introduced to refer to an external floating-point item that has USAGE DISPLAY.
- The term *external floating point* refers to both alphanumeric floating-point items and national floating-point items.

Changes in IBM Enterprise COBOL for z/OS, Version 3 Release 3

- XML support has been enhanced. A new statement, XML GENERATE, converts the content of COBOL data records to XML format. XML GENERATE creates XML documents encoded in Unicode UTF-16 or in one of several single-byte EBCDIC code pages.
- There are new and improved features of the Debug Tool:
 - Performance is improved when you use COBOL SYSDEBUG files.
 - You can more easily debug programs that use national data: When you display national data in a formatted dump or by using the Debug Tool LIST command, the data is automatically converted to EBCDIC representation using the code page specified in the CODEPAGE compiler option. You can use the Debug Tool MOVE command to assign values to national data items, and you can move national data items to or from group data items. You can use national data as a comparand in Debug Tool conditional commands such as IF or EVALUATE.
 - You can debug mixed COBOL-Java applications, COBOL class definitions, and COBOL programs that contain object-oriented syntax.

For further details about these enhancements to debugging support, see the *Debug Tool User's Guide*.

- DB2 Version 8 SQL features are supported when you use the integrated DB2 coprocessor.
- The syntax for specifying options in the COBJVMINTOPTIONS environment variable has changed.

Changes in IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 2

- The compiler has been enhanced to support new features of Debug Tool:
 - Playback support lets you record and replay application execution paths and data values.
 - Automonitor support displays the values of variables that are referenced in the current statement during debugging.
 - Programs that have been compiled with the OPTIMIZE and TEST(NONE,SYM,. . .) options are supported for debugging.

- The Debug Tool GOTO command is enabled for programs that have been compiled with the NOOPTIMIZE option and TEST option with any of its suboptions. (In earlier releases, the GOTO command was not supported for programs compiled with TEST(NONE, . . .).)

For further details about these enhancements to debugging support, see the *Debug Tool User's Guide*.

- Extending Java interoperability to IMS : Object-oriented COBOL programs can run in an IMS Java dependent region. The object-oriented COBOL and Java languages can be mixed in a single application.
- Enhanced support for Java interoperability:
 - The OPTIMIZE compiler option is fully supported for programs that contain OO syntax for Java interoperability.
 - Object references of type `objectArray` are supported for interoperation between COBOL and Java.
 - OO applications that begin with a COBOL main factory method can be invoked with the `java` command.
 - A new environment variable, `COBJVMINITOPTIONS`, is provided for initializing the Java virtual machine for OO applications that start with a COBOL program.
 - OO applications that begin with a COBOL program can, with some limitations, be bound as modules in a PDSE and run using batch JCL.
- Unicode enhancement for working with DB2: The code pages for host variables are handled implicitly when you use the DB2 integrated coprocessor. SQL DECLARE statements are necessary only for variables described with `USAGE DISPLAY` or `USAGE DISPLAY-1` when COBOL and DB2 code pages do not match.

Changes in IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 1

- Multithreading support: toleration of POSIX threads and signals, permitting applications with COBOL programs to run on multiple threads within a process
- Interoperation of COBOL and Java by means of object-oriented syntax, permitting COBOL programs to instantiate Java classes, invoke methods on Java objects, and define Java classes that can be instantiated in Java or COBOL and whose methods can be invoked in Java or COBOL
- Ability to call services provided by the Java Native Interface (JNI) to obtain additional Java capabilities, with a copybook `JNI.cpy` and special register `JNIENVPTR` to facilitate access
- Basic support for Unicode provided by `NATIONAL` data type and national (N, NX) literals, intrinsic functions `DISPLAY-OF` and `NATIONAL-OF` for character conversions, and compiler options `NSYMBOL` and `CODEPAGE`
 - Compiler option `CODEPAGE` to specify the code page used for encoding national literals, and alphanumeric and DBCS data items and literals
 - Compiler option `NSYMBOL` to control whether national or DBCS processing should be in effect for literals and data items that use the N symbol
- Basic XML support, including a high-speed XML parser that allows programs to consume inbound XML messages, verify that they are well formed, and transform their contents into COBOL data structures; with support for XML documents encoded in Unicode UTF-16 or several single-byte EBCDIC code pages

- Support for compilation of programs that contain CICS statements, without the need for a separate translation step
 - Compiler option CICS, enabling integrated CICS translation and specification of CICS options
- VALUE clauses for BINARY data items that permit numeric literals to have a value of magnitude up to the capacity of the native binary representation, rather than being limited to the value implied by the number of 9s in the PICTURE clause
- A 4-byte FUNCTION-POINTER data item that can contain the address of a COBOL or non-COBOL entry point, providing easier interoperability with C function pointers
- The following support is no longer provided (as documented in this *Migration Guide*):
 - SOM-based object-oriented syntax and services
 - Compiler options CMPR2, ANALYZE, FLAGMIG, TYPECHK, and IDLGEN
- Changed default values for the following compiler options: DBCS, FLAG(I,I), RENT, and XREF(FULL).

Changes in COBOL for OS/390 & VM, Version 2 Release 2

- Enhanced support for decimal data, raising the maximum number of decimal digits from 18 to 31 and providing an extended-precision mode for arithmetic calculations
- Enhanced production debugging using overlay hooks rather than compiled in hooks, with symbolic debugging information optionally in a separate file
- Support for compiling, linking, and running in the OS/390 UNIX System Services environment, with COBOL files able to reside in the hierarchical file system (HFS)
- Toleration of fork(), exec(), and spawn(); and the ability to call UNIX/POSIX functions
- Enhanced input-output function, permitting dynamic file allocation by means of an environment variable named in SELECT. . . ASSIGN, and the accessing of sequentially organized HFS files including by means of ACCEPT and DISPLAY
- Support for line-sequential file organization for accessing HFS files that contain text data, with records delimited by the new-line character
- COMP-5 data type, new to host COBOL, allowing values of magnitude up to the capacity of the native binary representation
- Significant performance improvement in processing binary data with the TRUNC(BIN) compiler option
- Support for linking of COBOL applications using the OS/390 DFSMS binder alone, with the prelinker required only in exceptional cases under CICS
- Diagnosis of moves (implicit or explicit) that result in numeric truncation enabled through compiler option DIAGTRUNC
- System-determined block size for the listing data set available by specifying BLKSIZE=0
- Limit on block size of QSAM tape files raised to 2 GB
- Support under CICS for DISPLAY to the system logical output device and ACCEPT for obtaining date and time
- Support for the DB2 coprocessor enabled through the SQL compiler option, eliminating the need for a separate precompile step and permitting SQL statements in nested programs and copybooks

- Support for the millennium language extensions now included in the base COBOL product

Changes in COBOL for OS/390 & VM V2 R1 Modification 2

- New compiler option ANALYZE to check the syntax of embedded SQL and CICS statements
- Extension of the ACCEPT statement to cover the recommendation in the Working Draft for Proposed Revision of ISO 1989:1985 Programming Language COBOL
- New intrinsic date functions to convert to dates with a four-digit year
- The millennium language extensions, enabling compiler-assisted date processing for dates containing two-digit and four-digit years

Requires IBM VisualAge® Millennium Language Extensions for OS/390 & VM (program number 5648-MLE) to be installed with your compiler.

Changes in COBOL for OS/390 & VM V2 R1 Modification 1

- Extensions to currency support for displaying financial data, including:
 - Support for currency signs of more than one character
 - Support for more than one type of currency sign in the same program
 - Support for the euro currency sign, as defined by the Economic and Monetary Union (EMU)

Changes in COBOL for OS/390 & VM, Version 2 Release 1

- Support has been added for dynamic link libraries (DLLs)
- Due to changes in the SOMobjects product that is delivered with OS/390 Release 3, changes in the JCL for building object-oriented COBOL applications were required.
- The INTDATE compiler option is no longer an installation option only. It can now be specified as an option when invoking the compiler.

How to send your comments

Your feedback is important in helping us to provide accurate, high-quality information. If you have comments about this information or any other Enterprise COBOL documentation, contact us in one of these ways:

- Use the Online Readers' Comment Form at www.ibm.com/software/awdtools/rcf/.
- Send your comments to the following address: compinfo@cn.ibm.com.

Be sure to include the name of the documentation, the publication number of the documentation, the version of Enterprise COBOL, and, if applicable, the specific location (for example, page number) of the text that you are commenting on.

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.

Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use software products successfully. The accessibility features in z/OS provide accessibility for Enterprise COBOL.

The major accessibility features in z/OS are:

- Interfaces that are commonly used by screen readers and screen-magnifier software
- Keyboard-only navigation
- Ability to customize display attributes such as color, contrast, and font size

Interface information

Assistive technology products work with the user interfaces that are found in z/OS. For specific guidance information, see the documentation for the assistive technology product that you use to access z/OS interfaces.

Keyboard navigation

Users can access z/OS user interfaces by using TSO/E or ISPF.

Users can also access z/OS services using IBM Rational Developer for System z.

For information about accessing these interfaces, see the following publications:

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Volume I*
- IBM Rational Developer for System z information centers

These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Accessibility of this information

The English-language XHTML format of this information that will be provided in the IBM System z[®] Enterprise Development Tools & Compilers Information Center at publib.boulder.ibm.com/infocenter/pdthelp/index.jsp is accessible to visually impaired individuals who use a screen reader.

To enable your screen reader to accurately read syntax diagrams, source code examples, and text that contains the period or comma PICTURE symbols, you must set the screen reader to speak all punctuation.

IBM and accessibility

See the Human Ability and Accessibility Center at www.ibm.com/able for more information about the commitment that IBM has to accessibility.

Part 1. Overview

Chapter 1. Introducing the new compiler and run time

This section provides an overview of the Enterprise COBOL compiler (IBM Enterprise COBOL for z/OS), and the common run time (Language Environment) and introduces you to the terminology used throughout this information.

Enterprise COBOL Version 5 executables are Program Objects and can reside only in PDSE data sets. If your COBOL load libraries are in PDS data sets, migrate them to PDSE data sets.

This manual assumes that you have completed your runtime migration to Language Environment. What does this mean? Briefly these are the conditions to be met before a COBOL runtime migration is complete:

1. The Language Environment dataset SCEERUN is installed in LNKLST or LPALST
2. There are no instances of COBLIB, VSCLLIB or COB2LIB in LNKLST or LPALST
3. There are no instances of COBLIB, VSCLLIB or COB2LIB in JCL STEPLIB or JOBLIB statements in batch jobs or in CICS startup JCL.
4. All statically bound runtime library routines for programs that are compiled with NORES have been REPLACed with routines from Language Environment.
5. IGZEBST bootstrap modules for VS COBOL II programs that are compiled with RES were either linked with the VS COBOL II runtime version of IGZEBST that has APAR PN74000 applied, or IGZEBST was REPLACed with IGZEBST from Language Environment.

If you understand these 4 conditions, and meet them all, you can skip to Chapter 4, “Planning to upgrade source programs,” on page 25.

If you understand these 4 conditions, but your shop has not completed its runtime library migration, you must complete that migration before using this book. You can use the *Enterprise COBOL V4.2 Compiler and Runtime Migration Guide* at <http://publibfp.dhe.ibm.com/epubs/pdf/igy3mg50.pdf> for help in completing your migration to Language Environment.

If you do not understand these conditions, then please continue reading these overview chapters. If you then discover that your shop has not completed its runtime library migration, use the *Enterprise COBOL V4.2 Compiler and Runtime Migration Guide* for help in completing your runtime library migration.

This section provides an overview of the Enterprise COBOL compiler (IBM Enterprise COBOL for z/OS), and the common run time (Language Environment) and introduces you to the terminology used throughout this information. This section includes the following information:

- Product relationships: compiler, run time, debug
- Comparison of COBOL compilers
- Language Environment's runtime support for different compilers
- Advantages of the new compiler and run time
- Suggestions for incremental migration
- Changes with the new compiler and run time

- General conversion tasks

Terminology clarification

In this information, we use the term Enterprise COBOL as a general reference to:

- IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 1
- IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
- IBM Enterprise COBOL for z/OS, Version 3 Release 3
- IBM Enterprise COBOL for z/OS, Version 3 Release 4
- IBM Enterprise COBOL for z/OS, Version 4 Release 1
- IBM Enterprise COBOL for z/OS, Version 4 Release 2
- IBM Enterprise COBOL for z/OS, Version 5 Release 1

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

- COBOL/370 Version 1 Release 1
- 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

See “Summary of changes to the COBOL compilers” on page xv for further details.

Product relationships: compiler, runtime library, debug

IBM Enterprise COBOL for z/OS is IBM's strategic COBOL compiler for the zSeries platform. Enterprise COBOL is comprised of features from IBM COBOL, VS COBOL II, and OS/VS COBOL with additional features such as multithread enablement, Unicode, XML capabilities, object-oriented COBOL syntax for Java interoperability, integrated CICS translator, and integrated DB2 coprocessor. Enterprise COBOL, as well as IBM COBOL and VS COBOL II, supports COBOL 85 Standard. Some features such as the CMPR2 compiler option and SOM-based object-oriented COBOL syntax that IBM COBOL supported are not available with Enterprise COBOL.

Language Environment provides a single language runtime library for COBOL, PL/I, C/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 runtime facilities, and a set of shared callable services. Language Environment is required to run Enterprise COBOL programs.

Debugging capabilities are provided by Debug Tool. Debug Tool provides significantly improved debugging function over previous COBOL debugging tools, and can be used to debug Enterprise COBOL programs, IBM COBOL programs, VS COBOL II programs running under Language Environment, and other programs including assembler, PL/I, and C/C++.

With OS/VS COBOL and VS COBOL II, the runtime library was included with the compiler. In addition, the debug component was also an optional part of a single COBOL product. In Enterprise COBOL Version 3 Debug Tool was included with the full-function version of the compiler.

With Enterprise COBOL Version 5, the compiler, the debugging component, and the runtime library are all separate, although the runtime library (Language Environment) is included with the z/OS operating system and does not need to be purchased separately.

Comparison of COBOL compilers

Table 4 gives an overview of the functions available with the latest releases of OS/VS COBOL, VS COBOL II, COBOL for MVS & VM, COBOL for OS/390 & VM, and shows the new functions available with the Enterprise COBOL compiler.

Table 4. Comparison of COBOL compilers

OS/VS COBOL	VS COBOL II	COBOL for MVS & VM	COBOL for OS/390 & VM	Enterprise COBOL for z/OS
				Support for: Java interoperability under IMS, OO support for Java interoperability, XML, integrated CICS translator, multithreading, Unicode
			Support for: DLLs 31 digits DB2 coprocessor OS/390 UNIX Enhanced support for Debug Tool	Support for: DLLs 31 digits DB2 coprocessor OS/390 UNIX Enhanced support for Debug Tool
		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	Extensions for: C interoperability, Intrinsic functions, Amendment to '85 Std, Support for: Language Environment Debug Tool
	COBOL 85 Standard, No intrinsic functions, Structured programming, DBCS National language, Improved CICS interface, 31-bit addressing, Reentrancy, Fast Sort Optimizer, Interactive debugging (full-screen mode)	COBOL 85 Standard, Structured programming, DBCS National language, Improved CICS interface, 31-bit addressing, Reentrancy, Fast Sort Optimizer, Interactive debugging (full-screen mode)	COBOL 85 Standard, Structured programming, DBCS National language, Improved CICS interface, 31-bit addressing, Reentrancy, Fast Sort Optimizer, Interactive debugging (full-screen mode)	COBOL 85 Standard, Structured programming, DBCS National language, Improved CICS interface, 31-bit addressing, Reentrancy, Fast Sort Optimizer, Interactive debugging (full-screen mode)

Table 4. Comparison of COBOL compilers (continued)

OS/VS COBOL	VS COBOL II	COBOL for MVS & VM	COBOL for OS/390 & VM	Enterprise COBOL for z/OS
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	COBOL 74 compatibility, 85 STD FIPS flagging, Dynamic loading, Batch debugging, Interactive debugging	COBOL 74 compatibility, 85 STD FIPS flagging, Dynamic loading, Batch debugging, Interactive debugging	85 STD FIPS flagging, Dynamic loading, Batch debugging, Interactive debugging

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

Language Environment's runtime support for different compilers

The OS/VS COBOL runtime library provided support for only OS/VS COBOL programs. Assembler programs could be included, but not VS COBOL II programs.

The VS COBOL II runtime library provided support for both OS/VS COBOL and VS COBOL II programs. Assembler programs could also be included.

Language Environment provides support for OS/VS COBOL programs, and VS COBOL II programs, as well as IBM COBOL and Enterprise COBOL programs. In addition, Language Environment provides support for other high-level languages, including PL/I, C/C++ and Fortran. Like its predecessors, assembler programs can be included in applications that run under Language Environment

Different versions of Enterprise COBOL have different minimum release level requirements for Language Environment. For example, Enterprise COBOL for z/OS, Version 4.2 required a minimum level of z/OS Version 1 Release 9 and Enterprise COBOL for z/OS, Version 5.1 requires a minimum level of z/OS Version 1 Release 13.

Advantages of the new compiler and run time

The Enterprise COBOL compiler and Language Environment run time provide additional functions over OS/VS COBOL, VS COBOL II, and IBM COBOL. Table 5 lists the advantages of the new compiler and run time and indicates whether they apply to VS COBOL II, OS/VS COBOL, IBM COBOL, or all three.

Table 5. Advantages of Enterprise COBOL and Language Environment

Advantage	Notes	Advantage over		
		OS/VS COBOL	VS COBOL II	IBM COBOL
XML support	Enterprise COBOL provides new statements for parsing and generating XML documents. These statements allow programs to transform XML content into COBOL data structures and COBOL data structures into XML documents.	X	X	X

Table 5. Advantages of Enterprise COBOL and Language Environment (continued)

Advantage	Notes	Advantage over		
		OS/VS COBOL	VS COBOL II	IBM COBOL
Java interoperation	Enterprise COBOL includes object-oriented COBOL syntax that enables COBOL to interoperate with Java. This Java interoperation is also supported under IMS.	X	X	X
Support to run in multiple threads	Enterprise COBOL has a toleration level of support for POSIX threads and signals. With Enterprise COBOL, an application can contain COBOL programs running on multiple threads within a process.	X	X	X
Support for Unicode	The COBOL Unicode support uses the product <i>z/OS Support for Unicode</i> .	X	X	X
Improved DB2 function	Enterprise COBOL includes support for DB2 stored procedures.	X	X	
	Support for the DB2 coprocessor	X	X	*
Improved CICS function	Enterprise COBOL includes CALL statement support (for faster CICS performance than when using EXEC CICS LINK) and eliminates the need for user-coded BLL cells. .	X		
	Increased WORKING-STORAGE space for DATA(24) and DATA(31) programs. For DATA(31), the limit is 2GB. For DATA(24), the limit is the available space below the 16-MB line.	X	X	X
	Support for the Integrated CICS translator	X	X	*
Usability enhancements	These enhancements include: <ul style="list-style-type: none"> • Large literals in VALUE clauses on COMP-5 items or BINARY items with TRUNC(BIN) • Function-pointer data type • Arguments specifying ADDRESS OF 	X	X	X
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.	X	X	
Above-the-line support	Virtual Storage Constraint Relief (VSCR) allows your programs to reside, compile, and access programs below or above the 16-MB line.	X		
	QSAM buffers can be above the 16-MB line for optimal support of DFSMS and data striping.	X	X	
	COBOL EXTERNAL data can now be above the line.	N/A	X	
31-digit support	Enterprise COBOL added support for numbers up to 31 digits when the ARITH(EXTEND) option is used.	X	X	*
z/OS UNIX system services support	The cob2 command can be used to compile and link COBOL programs in the z/OS UNIX shell. COBOL programs can call most of the C language functions defined in the POSIX standard.	X	X	

Table 5. Advantages of Enterprise COBOL and Language Environment (continued)

Advantage	Notes	Advantage over		
		OS/VS COBOL	VS COBOL II	IBM COBOL
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 Enterprise COBOL programs with Language Environment callable services to register the user-written condition handlers. Language Environment handles all condition management.	X	X	
High-precision math routines	Using Language Environment callable services, your programs can return the most accurate results.	X	X	
Support for multiple MVS tasks	RES applications can now execute independently under multiple MVS tasks. (For example, running two Enterprise COBOL programs at the same time from ISPF split screens.)	X	X	
Performance	Faster arithmetic computations	X		
	Faster dynamic and static CALL statements		X	
	Improved performance of variable-length MOVEs		X	
	Faster CICS performance if using the Language Environment CBLPSHPOP runtime option to prevent PUSH HANDLE and POP HANDLE for CALL statements.	X		
	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.	N/A	X	
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 statement. Additionally, under CICS, you can use the CALL statement to call PL/I or C programs in place of EXEC CICS LINK.	X	X	
Character manipulation	Improved bit and character manipulation using hex literals. Improved flexibility with character manipulation using reference modification	X		
Top-down modular program development	Support for top-down modular program development through nesting of programs and improved CALL and COPY functions	X		
Structured Programming Support	Support for structured programming coding through: <ul style="list-style-type: none"> • Inline PERFORM statements • The CONTINUE place-holder statement • The EVALUATE statement • Explicit scope terminators (for example: END-IF, END-PERFORM, END-READ) 	X		

Table 5. Advantages of Enterprise COBOL and Language Environment (continued)

Advantage	Notes	Advantage over		
		OS/VS COBOL	VS COBOL II	IBM COBOL
COBOL 85 Standard conformance	Support for COBOL 85 Standard	X		
	Support for Amendment 1 (Intrinsic Functions Module) of COBOL 85 Standard	X	X	
Subsystem support	Improved support for IMS, ISPF, DFSORT, DB2, WAS	X		
Support for reentrancy	All OS/VS COBOL programs are nonreentrant. Only reentrant programs can be loaded into shared storage (LPA or Shared Segments).	X		
Support for Debug Tool	Debug Tool provides the following benefits: <ul style="list-style-type: none"> • 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 that run on the host • Working in conjunction with Rational® Developer for System z, the ability to debug host programs from the workstation using a graphical user interface 	X	X	
	For COBOL for OS/390 & VM and later programs only: <ul style="list-style-type: none"> • Dynamic Debug feature which allows COBOL programs compiled without hooks to be debugged. 	X	X	
	For Enterprise COBOL Version 4 or later programs: <ul style="list-style-type: none"> • Compiler TEST suboption EJPD enables predictable GOTO/JUMPTO in programs also compiled with a non-zero OPTIMIZE level. Note: Unpredictable GOTO/JUMPTO in programs compiled with a non-zero OPTIMIZE level and TEST(NOJEPD) is available with the Debug Tool SET WARNING OFF command.	X	X	X
Runtime options	ABTERMENC and TERMTHDACT- allow you to control error behavior.	X	X	
	CBLQDA - allows you to control dynamic allocation of QSAM files.		X	
	LANGUAGE - allows you to change language of runtime error messages.	X		
	RPTSTG - allows you to obtain storage usage reports.	X		
	Storage options - allow you to control where storage is obtained and the amount of storage used.	X	X	

Table 5. Advantages of Enterprise COBOL and Language Environment (continued)

Advantage	Notes	Advantage over		
		OS/VS COBOL	VS COBOL II	IBM COBOL
Compiler options for Enterprise COBOL Version 5	<p>There have been many changes to compiler options and suboptions for Enterprise COBOL Version 5. For details about those changes, see “Compiler option changes in IBM Enterprise COBOL for z/OS, Version 5” on page 181. The following compiler options are available to Enterprise COBOL Version 5 programs only:</p> <ul style="list-style-type: none"> • AFP(VOLATILE NOVOLATILE) • ARCH(<i>n</i>) • DISPSIGN(SEP COMPAT) • HGPR(PRESERVE NOPRESERVE) • MAXPCF(<i>n</i>) • STGOPT NOSTGOPT 	X	X	X
Compiler options for Enterprise COBOL Version 4	<p>The following compiler options are available to Enterprise COBOL Version 4 programs only:</p> <ul style="list-style-type: none"> • XMLPARSE - controls whether the z/OS XML System Services parser or the existing COBOL parser is used for XML PARSE statements. With the XMLPARSE(COMPAT) option, XML parsing is compatible with Enterprise COBOL Version 3. With the XMLPARSE(XMLSS) options, the z/OS System Services parser is used and new XML parsing capabilities are enabled. • OPTFILE - controls whether compiler options are read from a data set specified in a SYSOPTF DD statement. • SQLCCSID - controls coordination of the coded character set ID (CCSID) between COBOL and DB2. • BLOCK0 - activates an implicit BLOCK CONTAINS 0 clause for all eligible QSAM files in a program. • MSGEXIT - The MSGEXIT suboption of the EXIT compiler option provides a facility for customizing compiler messages (changing their severity or suppressing them), including FIPS (FLAGSTD) messages. 	X	X	X

Table 5. Advantages of Enterprise COBOL and Language Environment (continued)

Advantage	Notes	Advantage over		
		OS/VS COBOL	VS COBOL II	IBM COBOL
Compiler options for Enterprise COBOL Version 3	<p>The following compiler options are available to Enterprise COBOL Version 3 and later programs only:</p> <ul style="list-style-type: none"> • CICS - enables the integrated CICS translator capability and specifies CICS options. NOCICS is the default. • CODEPAGE - specifies the code page used for encoding contents of alphanumeric and DBCS data items at run time as well as alphanumeric, national, and DBCS literals in a COBOL source program. • MDECK(COMPILE, NOCOMPILE) - controls whether output from library processing is written to a file and whether compilation continues normally after library processing and the generation of the output file. • NSYMBOL(NATIONAL, DBCS) - controls the interpretation of the "N" symbol used in literals and picture clauses, indicating whether national or DBCS processing is assumed. • THREAD - indicates that the COBOL program is to be enabled for execution in a Language Environment enclave with multiple POSIX threads or PL/I tasks. The default is NOTHREAD. 	X	X	X
Compiler options for COBOL for OS/390 & VM	<p>The following compiler options are available to COBOL for OS/390 & VM and later programs only:</p> <ul style="list-style-type: none"> • 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. 	X	X	
Compiler options for COBOL for MVS & VM	<p>The following compiler options are available to COBOL for MVS & VM and later programs:</p> <ul style="list-style-type: none"> • 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 runtime 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 16-MB line. 	X	X	
* The integrated DB2 coprocessor, integrated CICS translator, and 31-digit support were added as new features to COBOL for OS/390 & VM, Version 2 Release 2.				

Changes with the new compiler and run time

With Enterprise COBOL, you may find that recompiling existing COBOL applications is affected by several areas such as the removal of compiler options, different default compiler options, unsupported SOM-based OO COBOL, and an integrated DB2 coprocessor, and an integrated CICS translator. The following information is a brief description of the removed or improved element and the actions required to ensure compatibility.

CMPR2 compiler option not available

Enterprise COBOL does not provide the CMPR2 compiler option. Existing programs compiled with CMPR2 must be converted to NOCMPR2 (COBOL 85 Standard) in order to compile them with Enterprise COBOL.

For additional details, see:

- Chapter 5, “Upgrading OS/VS COBOL source programs,” on page 43
- Chapter 7, “Upgrading VS COBOL II source programs,” on page 93
- Chapter 9, “Upgrading IBM COBOL source programs,” on page 101

FLAGMIG compiler option

Enterprise COBOL V5 does not provide the FLAGMIG compiler option.

To aid you with migration to Enterprise COBOL V5, there is a new option in Enterprise COBOL V4.2, FLAGMIG4, to flag source code syntax-related changes required to move to Enterprise COBOL V5.

For additional details about the FLAGMIG option, see:

- Chapter 5, “Upgrading OS/VS COBOL source programs,” on page 43
- Chapter 7, “Upgrading VS COBOL II source programs,” on page 93
- Chapter 9, “Upgrading IBM COBOL source programs,” on page 101

SOM-based object-oriented COBOL not available

Enterprise COBOL does not support SOM-based OO COBOL; however, Enterprise COBOL provides OO syntax to facilitate the interoperation of COBOL and Java programs. The removal of SOM-based OO COBOL from Enterprise COBOL included the removal of the compiler options TYPECHK and IDLGEN because they require SOM to run. Applications utilizing SOM-based OO COBOL must be redesigned to upgrade to Java-based OO COBOL syntax or redesigned as procedural (non-OO) COBOL.

For additional details and compatibility considerations, see “Upgrading SOM-based object-oriented (OO) COBOL programs” on page 140.

Integrated DB2 coprocessor available

Enterprise COBOL provides an integrated DB2 coprocessor that allows the Enterprise COBOL compiler to handle both native COBOL statements and embedded SQL statements in a source program. You can choose to migrate from the separate DB2 precompiler to the integrated DB2 coprocessor, or you can choose to continue using the separate DB2 precompiler.

The SQL compiler option must be specified to enable the DB2 coprocessor to process a COBOL source program that contains SQL statements.

For additional details and compatibility considerations, see:

- Chapter 19, “DB2 coprocessor conversion considerations,” on page 217

Integrated CICS translator available

Enterprise COBOL provides an integrated CICS translator that allows the Enterprise COBOL compiler to handle both native COBOL statements and embedded CICS statements in a source program. You can choose to migrate from the separate CICS translator to the integrated CICS translator, or to continue using the separate CICS translator.

The CICS compiler option must be specified to enable the CICS translator to process a COBOL source program that contains CICS statements.

For additional details and compatibility considerations, see:

- Chapter 18, “CICS conversion considerations for COBOL source,” on page 211

General migration tasks

Depending on your shop's programming environment, you will likely have to complete one or more migration tasks to move to the new compiler and run time.

These tasks include:

- Planning your strategy
- Upgrading your source to Enterprise COBOL
- Adding Enterprise COBOL programs to existing applications

Planning your strategy

Before upgrading your source programs to Enterprise COBOL, develop a conversion strategy. For help in completing your runtime library migration to Language Environment, see the *Enterprise COBOL V4.2 Compiler and Runtime Migration Guide* at <http://publibfp.dhe.ibm.com/epubs/pdf/igy3mg50.pdf>

Your migration strategy might be to gradually recompile entire existing applications with Enterprise COBOL as needed. You may also decide to recompile individual programs as you go.

Upgrading your source to Enterprise COBOL

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

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 Enterprise COBOL. You should use conversion tools to aid in this conversion. For details, see “Converting to COBOL 85 Standard” on page 54.

VS COBOL II

From a conversion standpoint, VS COBOL II and Enterprise COBOL Version 5 have the following language differences:

- Removal of CMPR2 support
- Behavior of some SEARCH ALL statements
- New reserved words
- Simplified TEST compiler option
- Removal of runtime support for SIMVRD
- Removal of support for the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and the syntax: GO TO MORE-LABELS.

A complete list of reserved words, including those reserved for object-oriented COBOL is included in Appendix B, “COBOL reserved word comparison,” on page 233.

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 Enterprise COBOL without change and receive the same results. For details, see Chapter 7, “Upgrading VS COBOL II source programs,” on page 93.

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. The CMPR2 compiler option is not supported by Enterprise COBOL, requiring source conversion for all VS COBOL II Release 1 or 2 programs as well as any VS COBOL II Release 3 or 4 programs that were compiled with CMPR2. Conversion tools can help you upgrade your source programs to COBOL 85 Standard. Details of language differences between CMPR2 and NOCMPR2 are included in “Migrating from the CMPR2 compiler option to NOCMPR2” on page 107.

For details about the conversion tools available to upgrade source programs, see Appendix C, “Conversion tools for source programs,” on page 249.

IBM COBOL

Many IBM COBOL programs will compile without change under Enterprise COBOL.

The following programs, however, will need to be upgraded before compiling with Enterprise COBOL:

- Programs compiled with the CMPR2 compiler option
- Programs that have SOM-based object-oriented COBOL syntax
- Programs that use words which are now reserved in Enterprise COBOL
- Programs that have undocumented IBM COBOL extensions
- Programs that contain the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and optionally the syntax: GO TO MORE-LABELS.

For details, see Chapter 9, “Upgrading IBM COBOL source programs,” on page 101.

Enterprise COBOL Version 3

Most Enterprise COBOL Version 3 programs will compile without change under Enterprise COBOL Version 5.

The following programs, however, will need to be upgraded:

- Programs that use words which are now reserved in Enterprise COBOL

- Programs that contain the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and the syntax: GO TO MORE-LABELS.
- Programs that contain XML PARSE statements.

For details, see Chapter 11, “Upgrading programs from Enterprise COBOL Version 3,” on page 147.

Enterprise COBOL Version 4

Most Enterprise COBOL Version 4 programs will compile without change under Enterprise COBOL Version 5.

The following programs, however, will need to be upgraded:

- Programs that use words which are now reserved in Enterprise COBOL
- Programs that contain the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and the syntax: GO TO MORE-LABELS.
- Programs that contain XML PARSE statements and were compiled with the XMLPARSE(COMPAT) compiler option.

For details, see Chapter 13, “Upgrading from Enterprise COBOL Version 4,” on page 165.

Adding Enterprise COBOL programs to existing applications

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

Note: You should use this Migration Guide only if you complete your runtime migration from pre-LE runtime libraries to Language Environment. The following conditions have to be met before a COBOL runtime migration is complete:

- The Language Environment data set SCEERUN is installed in LNKLIST or LPALST.
- There are no instances of COBLIB, VSCLLIB, or COB2LIB in LNKLIST or LPALST.
- There are no instances of COBLIB, VSCLLIB, or COB2LIB in JCL STEPLIB or JOBLIB statements in batch jobs or in CICS startup JCL.
- All statically bound runtime library routines for programs that are compiled with NORES have been REPLACed with routines from Language Environment.
- IGZEBST bootstrap modules for VS COBOL II programs that are compiled with RES were either linked with the VS COBOL II runtime version of IGZEBST that has APAR PN74000 applied, or IGZEBST was REPLACed with IGZEBST from Language Environment.

If these steps have not been completed, please first complete all runtime migration activities in the *Enterprise COBOL Version 4.2 Compiler and Runtime Migration Guide* prior to following the steps here.

When adding Enterprise COBOL programs to existing applications, you must be aware of the following items:

- Restrictions of running programs with certain old COBOL programs
- Acquiring WORKING-STORAGE both above and below the 16-MB line
- Effect of compiler option changes
- Reserved word changes
- Other behavior differences with Enterprise COBOL V5

| For details, see Chapter 16, “Adding Enterprise COBOL V5.1 programs to existing
| COBOL applications,” on page 197.

Restriction: You cannot mix Enterprise COBOL Version 5 with:

- OS/VS COBOL programs. You must migrate them to Enterprise COBOL.
- VS COBOL II NORES programs. You must migrate them to Enterprise COBOL.

Chapter 2. Do I need to recompile?

Ideally, programs should be compiled with a supported compiler (currently only IBM Enterprise COBOL for z/OS is supported) and run with a supported runtime library (Language Environment). You can migrate programs gradually, in two stages:

- Stage 1: Runtime migration. You can use the *Enterprise COBOL V4.2 Compiler and Runtime Migration Guide* at <http://publibfp.dhe.ibm.com/epubs/pdf/igy3mg50.pdf> for help in completing your runtime library migration.
- Stage 2: Compiler migration (you may compile only one or many programs in existing applications)

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

Migration basics

The migration process involves runtime migration (moving your applications to a new runtime library) and source migration (upgrading your source programs). As part of the migration process, you will also need to do inventory assessment and testing. As stated previously, you are not required to do your runtime migration and recompilation concurrently.

For more details about the migration process, see “General migration tasks” on page 13.

Runtime migration

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

Moving to Language Environment

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

Note: If your IGZEBST bootstrap routine from VS COBOL II has PN74000 installed, you do not need to REPLACE this IGZEBST with the Language Environment version of IGZEBST.

If the programs are compiled with the RES option, make the Language Environment library routines available at run time in place of the OS/VS COBOL or VS COBOL II library routines by using LNKLST, LPALST, JOBLIB, or STEPLIB.

Do not make more than one COBOL runtime library available to your applications at run time. For example, there should be one and only one COBOL runtime 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. In addition, if you have more than one runtime library in your concatenation, then you have an invalid configuration that is not supported by IBM.

If you have not yet completed your runtime library migration, you must complete that migration before using this book. You can use the *Enterprise COBOL V4.2 Compiler and Runtime Migration Guide* at <http://publibfp.dhe.ibm.com/epubs/pdf/igy3mg50.pdf> for help in completing your runtime library migration.

Source migration

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

Source migration and recompilation is required for OS/VS COBOL programs and VS COBOL II NORES programs if they are to be called by (or need to call) Enterprise COBOL Version 5 programs. Enterprise COBOL V5 programs can dynamically call (and be dynamically called by) VS COBOL II RES programs.

Source migration usually consists of upgrading the source language level that is used (such as from Standard COBOL 74 supported by OS/VS COBOL to COBOL 85 Standard supported by Enterprise COBOL). 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 249.

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

In some cases IBM will continue to provide support for OS/VS COBOL and VS COBOL II programs that run under Language Environment.

IBM will continue to provide service support for the running of programs compiled with the OS/VS COBOL Release 2 and VS COBOL II Release 3 and higher compilers when these programs use the Language Environment runtime library versions of the COBOL library routines with the following exceptions:

- OS/VS COBOL programs running under CICS Transaction Server
- OS/VS COBOL programs interoperating with Enterprise COBOL V5 programs
- VS COBOL II programs compiled with the NORES option interoperating with Enterprise COBOL V5 programs

For example, the library routines for OS/VS COBOL programs exist in the OS/VS COBOL, the VS COBOL II, and the Language Environment runtime libraries. OS/VS COBOL programs running with the OS/VS COBOL runtime library or the VS COBOL II runtime library are not supported by IBM Service. If your OS/VS COBOL programs are running using a supported release of the Language Environment runtime library, your programs are supported by IBM Service but they cannot interoperate with Enterprise COBOL V5 programs. .

In CICS TS (Transaction Server), you can no longer run OS/VS COBOL programs.

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 and not interoperating with Enterprise COBOL V5 programs. Once you have migrated your runtime library to Language Environment, you can run your source code through a source conversion tool, such as the COBOL and CICS Conversion Aid (CCCA) and then compile using the Enterprise COBOL compiler.

For more information about CCCA, see Appendix C, “Conversion tools for source programs,” on page 249.

Interoperability with older levels of IBM COBOL programs

There are some restrictions for Enterprise COBOL V5 programs to call or be called by (interoperate) with programs compiled with earlier versions of COBOL.

Enterprise COBOL V5 programs cannot interoperate with OS/VS COBOL or VS COBOL II NORES programs in a single application. A COBOL run unit (Language Environment enclave) that contains an Enterprise COBOL V5 compiled program must not contain any OS/VS COBOL or VS COBOL II NORES programs.

Note: Run units that contain only COBOL programs compiled with Enterprise COBOL V4 or earlier versions can interoperate with OS/VS COBOL and VS COBOL II NORES programs.

Programs compiled with Enterprise COBOL V5 can interoperate with programs compiled with VS COBOL II or later, based on the following conditions and CALL types:

- Static calls. Enterprise COBOL V5 compiled programs can be bound or link-edited with the following object modules or programs to form a single program object. The programs within the program object can specify static calls to and from each other.
 - Programs that are compiled with VS COBOL II with the RES compiler option
 - Programs that are compiled with any IBM COBOL compiler versions subsequent to VS COBOL II
 - Programs that are compiled with Enterprise COBOL V3 or V4

Note: Programs that are compiled with VS COBOL II with the NORES compiler option specified cannot interoperate with programs compiled with Enterprise COBOL V5.

- Dynamic calls. Program modules that contain programs compiled with VS COBOL II with the RES option, or subsequent versions of COBOL can also interoperate with Enterprise COBOL V5 program objects by using dynamic CALL statements.
- DLL calls. Program modules that are compiled with earlier versions of COBOL that supported DLL linkage can interoperate with Enterprise COBOL V5 program objects by using DLL linkage.

Part 2. Migration strategies

Chapter 3. Compiler upgrade checklist

To upgrade your programs to Enterprise COBOL, use the following checklist.

Do these tasks:

1. If your COBOL load libraries are in PDS data sets, migrate them to PDSE data sets.
2. Complete runtime migration, which means:
 - The Language Environment dataset SCEERUN is installed in LNKST or LPALST
 - There are no instances of COBLIB, VSCLLIB or COB2LIB in LNKST, or LPALST
 - There are no instances of COBLIB, VSCLLIB or COB2LIB in JCL STEPLIB or JOBLIB statements in batch jobs or in CICS startup JCL
 - All statically bound runtime library routines for programs that are compiled with NORES have been REPLACEd with routines from Language Environment.
 - IGZEBST bootstrap modules for VS COBOL II programs that are compiled with RES were either linked with the VS COBOL II runtime version of IGZEBST that has APAR PN74000 applied, or IGZEBST was REPLACEd with IGZEBST from Language Environment.
3. Ensure that all software and hardware prerequisites as defined in the *Licensed Program Specifications* Enterprise COBOL are satisfied. (Get the *Licensed Program Specifications* from the Enterprise COBOL for z/OS library at <http://www.ibm.com/support/docview.wss?uid=swg27036733>.)
4. Install prerequisite PTFs for the Language Environment runtime library on all systems where COBOL programs might be compiled or run, including on all production systems.
5. Ensure that all systems on which COBOL will run, and all software that needs to work with COBOL (for example z/OS, Debug Tool, Fault Analyzer, and DB2), are ready for programs compiled with the new COBOL compiler. For a list of APARs, see “Prerequisite software and service for Enterprise COBOL V5” on page 179
6. Save the old COBOL compiler for emergency use.
7. Purchase and install the new Enterprise COBOL compiler.
8. Set up the default compiler options and your library control system options for the new compiler to be compatible with the old compiler. For future reuse, document any customization or set up that you do.
9. Depending on which COBOL compiler you are migrating from, you might need to make COBOL source-code changes. For details, see the topic in the *Upgrading programs* section of this information which applies to your current compiler.
10. Follow the recommended COBOL compiler migration strategy, which takes advantage of existing application development processes:
 - Whenever you make code changes, compile using the new compiler.
 - You can mix programs that were compiled with the old and new compiler within an application, with restrictions. It is often not necessary to

recompile all programs. If you do not recompile all programs you will have to maintain older compilers for the programs that have not yet been recompiled.

The benefit of this strategy is that because developers normally test their changes, the compiler migration will not require extra testing, so in effect will be free!

11. Develop a strategy for recompiling programs that aren't going to be changed in the course of normal development, for example:

- Recompile all programs in a module if any program in the module is changed.
- Recompile all programs in an application if any program in the application is changed.
- Schedule recompilation of all programs application by application.

Note that running old modules under Language Environment is supported.

12. After all programs have been compiled with the new compiler, uninstall the old compiler. That way, you save license fees!

Chapter 4. Planning to upgrade source programs

You can follow a general strategy for upgrading source programs to Enterprise COBOL.

The following tasks are necessary, and should be performed in roughly the following order:

1. Preparing to upgrade your source
2. Taking an inventory of your applications
3. Prioritizing your applications
4. Setting up a conversion procedure
5. Making application program updates

Because of 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, at some future date the older compilers and any supported fixes will not be available. At that point, you will be forced to do a 'quick' migration, and this might be at a very inconvenient time.

Preparing to upgrade your source

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

- Installing Enterprise COBOL
- Assessing storage requirements
- Deciding which conversion tools to use
- Educating your programmers on new compiler features

Installing Enterprise COBOL

If you haven't already done so, install the compiler; see the *Program Directory for Enterprise COBOL*.

Assessing storage requirements

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

During conversion, you will need DASD storage for your current COBOL compilers as well as for the Enterprise COBOL compiler. When you have completed conversion, and if you have upgraded all of your OS/VS COBOL, VS COBOL II, or IBM COBOL programs to Enterprise COBOL, you will be able to free the storage reserved for your current COBOL compiler.

The program object produced from the same source code when compiled with Enterprise COBOL V5 will be larger than the load module produced when compiled with all earlier versions of COBOL.

Deciding 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 Enterprise COBOL programs:

COBOL Conversion Tool (CCCA)

The COBOL and CICS/VS Command Level Conversion Aid (CCCA) automatically converts your old COBOL programs, either OS/VS COBOL, VS COBOL II, or IBM COBOL with CMPR2, into COBOL 85 Standard code that you can compile with Enterprise COBOL. It also provides you with reports of the statements that were changed. CCCA is included with the IBM Debug Tool product.

For more information about CCCA, see Appendix C, “Conversion tools for source programs,” on page 249.

OS/VS COBOL MIGR compiler option

The MIGR option identifies source statements that need to be converted to compile under Enterprise COBOL.

CMPR2, FLAGMIG, and NOCOMPILE compiler options

The COBOL CMPR2, FLAGMIG, and NOCOMPILE options identify source statements that need to be converted to compile under Enterprise COBOL. The CMPR2 and FLAGMIG options are not available in Enterprise COBOL, but you can use your older compilers with these options to flag the statements that need to be changed in order to compile with Enterprise COBOL.

Enterprise COBOL V4.2 FLAGMIG4 compiler option

A new compiler option, FLAGMIG4, is available with APAR PM93450 for Enterprise COBOL V4.2 to help you migrate to Enterprise COBOL V5. The FLAGMIG4 option identifies language elements in Enterprise COBOL V4 programs that are not supported, or that are supported differently in Enterprise COBOL V5. The compiler generates a warning diagnostic message for all such language elements.

Another conversion tool you might want to use is COBOL Report Writer Precompiler. It 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.

These conversion tools are fully described in Appendix C, “Conversion tools for source programs,” on page 249.

If you plan to use CCCA 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.

Educating your programmers on new compiler features

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

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

For information about Enterprise COBOL and Language Environment education available through IBM, you can call 1-800-IBM-TEACH (1-800-426-8322). You can also get information directly from Language Environment publications or technical conferences such as SHARE, www.share.org.

After your programmers are familiar with Enterprise COBOL features, they can assist you in taking the inventory of programs as described in “Taking an inventory of your applications.”

Taking an inventory of your applications

In planning the upgrade to Enterprise COBOL, you need to take a comprehensive inventory of applications in which you have programs that you intend to compile with Enterprise COBOL.

The Debug Tool Load Module Analyzer can determine the language translator that was used for each object in your load modules. See “Debug Tool Load Module Analyzer” on page 256 for more information.

The Edge Portfolio Analyzer can provide assistance in taking an inventory of your existing load modules by reporting the compiler, compiler release, and compiler options used. See “The Edge Portfolio Analyzer” on page 256 for more information.

Language Environment can help you find out whether you are ever running OS/VS COBOL programs from your inventory. Install the fix for APAR PM86742 to your Language Environment and look for one of these warning messages about detected OS/VS COBOL programs at run time:

IGZ0268W

An invocation was made of OS/VS COBOL program "program-name".

IGZ0269W

"program-lang" version "program-version" program "program-name" made a call to OS/VS COBOL program "program-name".

Rational Asset Analyzer for z/OS can aid by analyzing the impact of a code change for an application. See “Rational Asset Analyzer” on page 253 for more information.

Taking an inventory of vendor tools, packages, and products

Before you can begin upgrading your source, you must know whether your vendor tools, packages, and products are designed to work with Enterprise COBOL. Verify:

- COBOL code generators generate COBOL 85 Standard programs that can be compiled with Enterprise COBOL.
- COBOL packages are written in COBOL 85 Standard language that can be compiled with Enterprise COBOL.
- Third-party tools such as debuggers and databases support Enterprise COBOL.

Taking an inventory of COBOL applications

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

For all previous versions of COBOL:

- Programmer responsible
- COBOL Standard level of source program (68, 74, 85)
- Compiler used (ANS COBOL V4, OS/VS COBOL, VS COBOL II, IBM COBOL, Enterprise COBOL V3, Enterprise COBOL V4)
- Compiler options used, especially CMPR2, NORES, XMLPARSE
- 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
- Use of SIMVRDS, SOM-based OO, Millennium Language Extensions, or LABEL declaratives

This information is useful to you in the next step of your planning task, "Prioritizing your applications."

Prioritizing your applications

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

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.

Assigning complexity ratings

Complexity ratings are defined based on the effort required to convert, test, and coordinate a construct or program. The ratings used in Table 6 on page 29 are defined as:

Complexity rating	Requirement
0	All code converted by CCCA without error; code compiles correctly under Enterprise COBOL
1-3	Most code converted without error by CCCA 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

Complexity rating	Requirement
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 that you assign for any program within the application is the complexity rating for the entire application.

Table 6 shows estimated complexity ratings for conversions of specific program attributes.

Table 6. Complexity ratings for program 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) ¹		4		
VS COBOL II or later compiled with CMPR2	Compiler option CMPR2 not supported	1	C	
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 ²		
	Module does not need to be upgraded	6		
CICS macro level program		10		
Compiled by Full ANS COBOL V4 compiler (pre- 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

Table 6. Complexity ratings for program attribute conversions (continued)

Program attribute	Description of attribute	Complexity rating		
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		
Access methods used	ISAM ³	10	M	C
	BDAM	10	C ⁴	
	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		
SIMVRD		3		
SOM-based OO		8		
LABEL declaratives		3		
XMLPARSE(COMPAT)		7		

1. For additional information, see Appendix G, “Preventing file status 39 for QSAM files,” on page 283.

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

3. Support for ISAM was removed with z/OS 1.7.

4. This is a partial conversion.

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).

Determining conversion priority

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

Table 7 shows one method of relating program complexity ratings to conversion priorities. (The highest priority is “1” and the lowest priority is “6”.)

Table 7. 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

Table 7. Assigning program conversion priorities (continued)

Conversion priority	Complexity rating	Other considerations
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 situations when deciding on conversion priorities:

- If your application is at the limits of the storage available below the 16-MB line, it is a prime candidate for conversion to Enterprise COBOL. With z/OS architecture you can obtain virtual storage constraint relief.

After you determine the priority of each program that 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.

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 Enterprise COBOL V5.1 programs to existing COBOL applications,” on page 197.

Setting 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 Enterprise COBOL program, do the steps shown in the flowchart below.

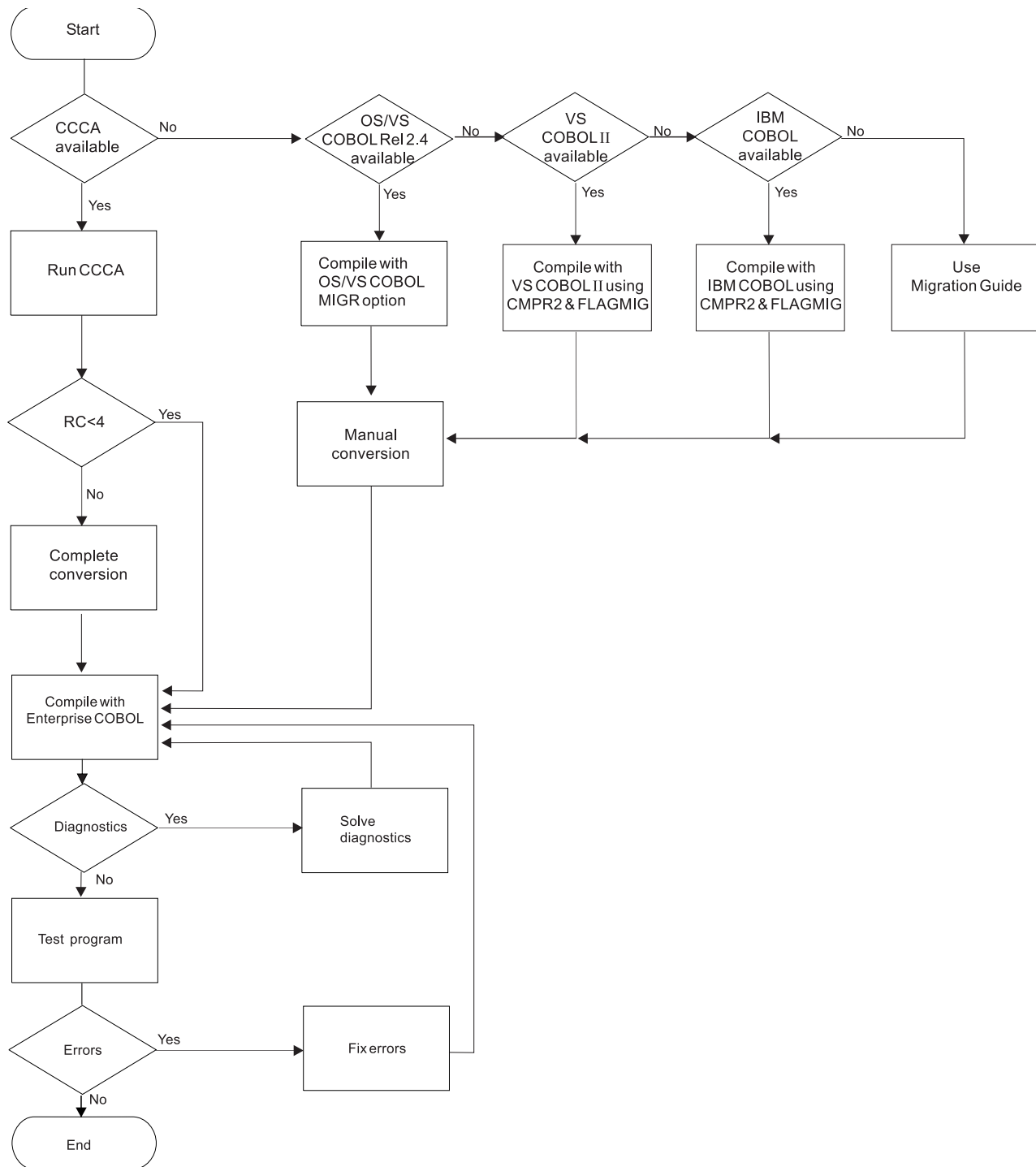


Figure 1. Steps for converting an OS/VS COBOL program to an Enterprise COBOL program

Programs with CICS

To convert an OS/VS COBOL program that contains CICS commands to an Enterprise COBOL program, do the steps shown in the flowchart below.

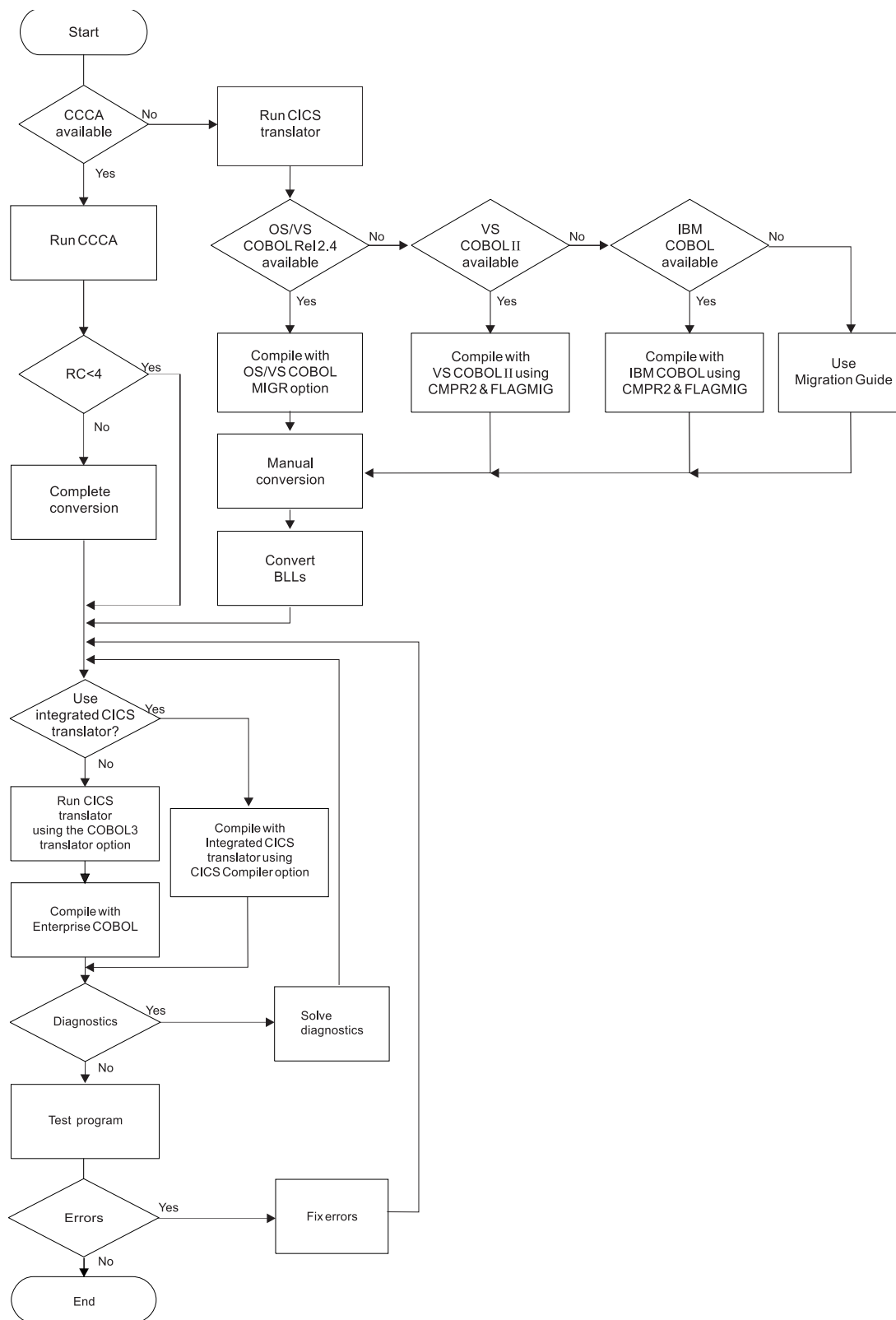


Figure 2. Steps for converting an OS/VS COBOL program containing CICS commands

Programs with Report Writer statements to be discarded

To convert an OS/VS COBOL program with Report Writer statements to Enterprise COBOL, and remove Report Writer statements, perform the steps shown in the flowchart below.

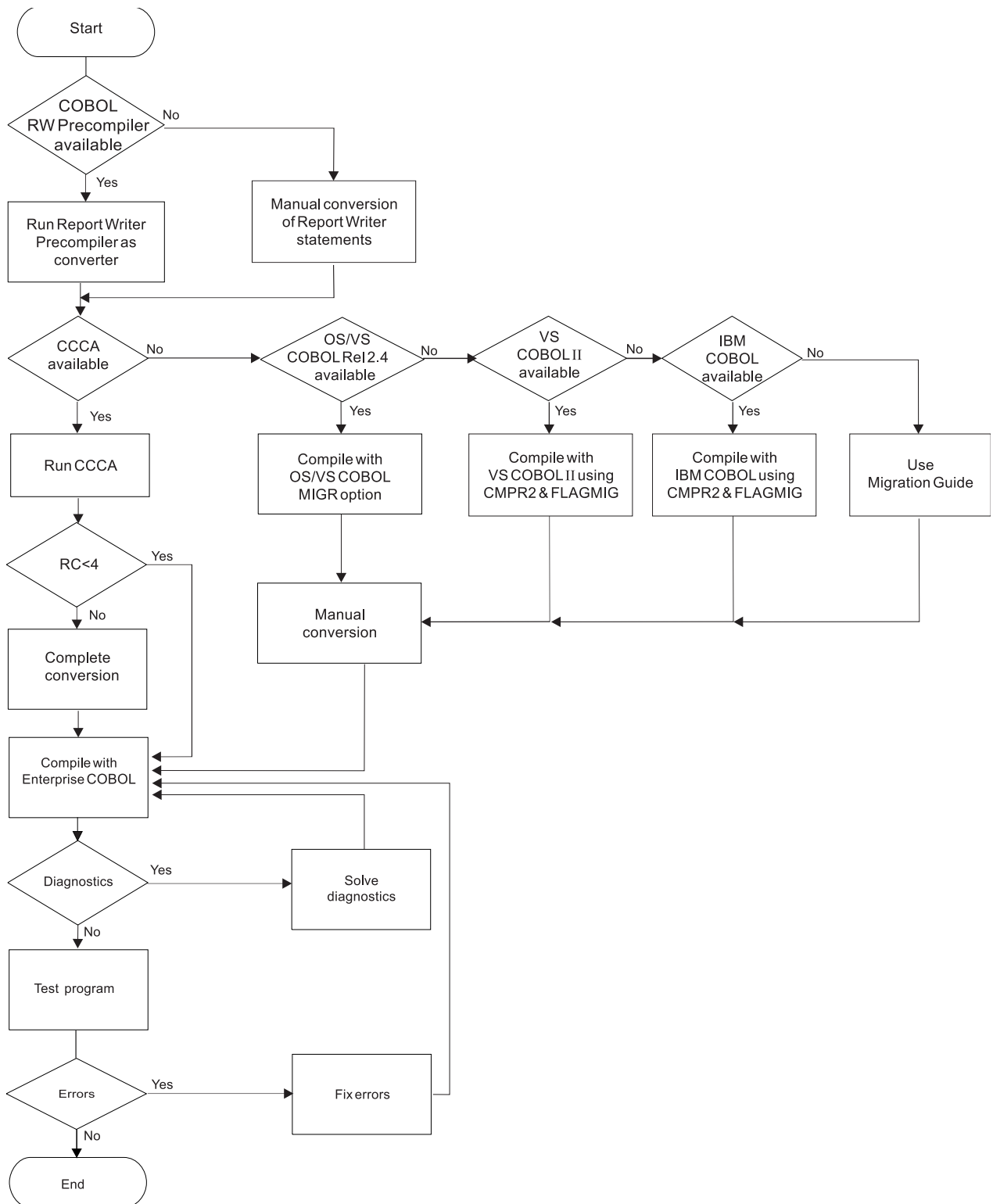


Figure 3. Steps for converting an OS/VS COBOL program and discarding Report Writer statements

Programs with Report Writer statements to be retained

To convert an OS/VS COBOL program that contains Report Writer statements to an Enterprise COBOL program, and retain the Report Writer statements in the

source code, do the steps shown in the flowchart below.

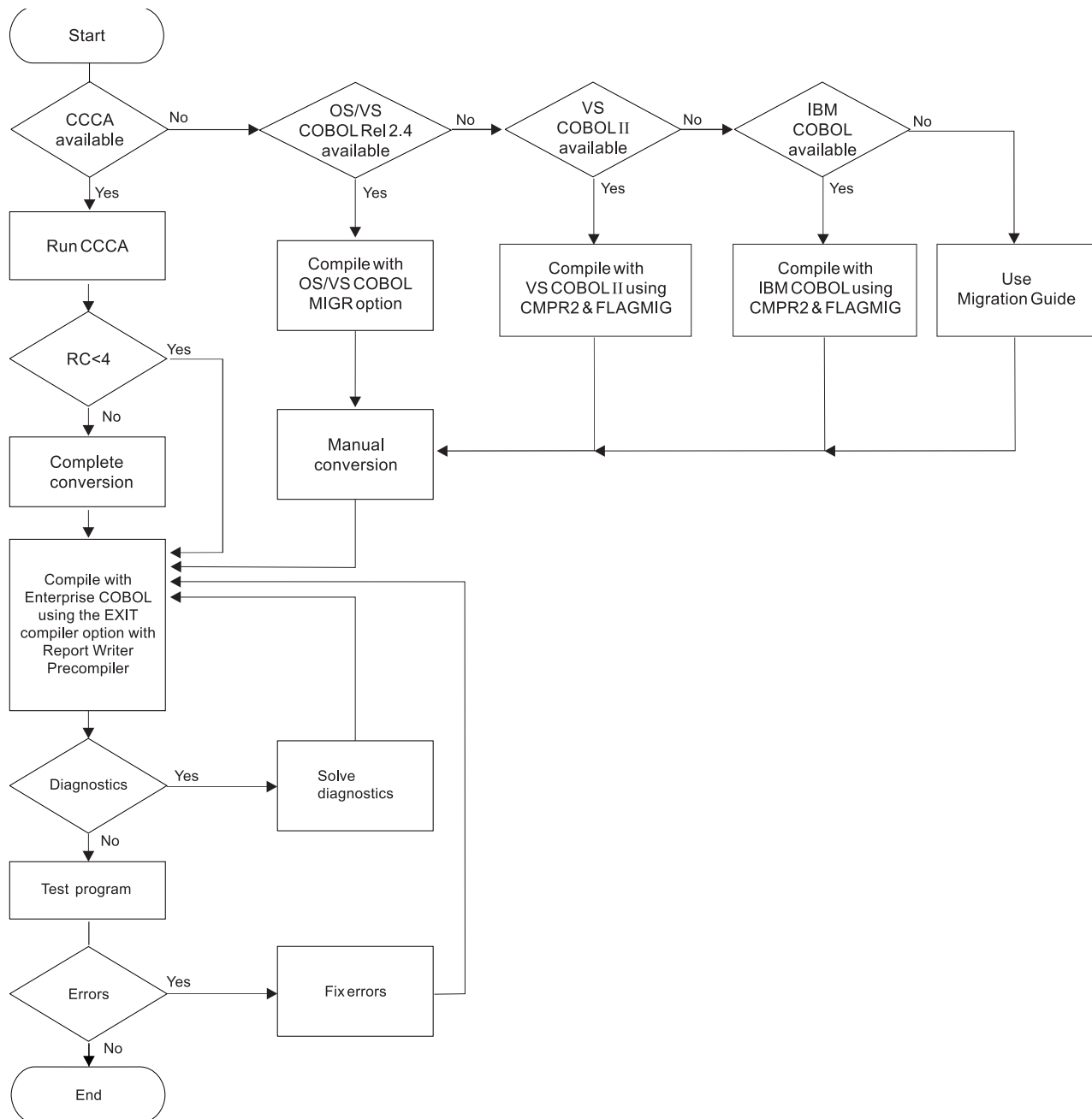


Figure 4. Steps for converting an OS/VS COBOL program and retaining Report Writer statements

Making 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 backup (a benchmark to compare to and a version to which to recover if the converted modules have problems).

1. Update the 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 the available source code.

Whenever possible, use the conversion tools described in Appendix C, “Conversion tools for source programs,” on page 249. Otherwise, update the source code manually.

3. Compile, link-edit, and run.

After the source has been updated, you can process the program as you would a newly written Enterprise COBOL program.

4. Debug.

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

5. Test the converted programs

After upgrading your source to Enterprise 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 G, “Preventing file status 39 for QSAM files,” on page 283.
- Performance differences.
- Sign handling problems—S0C7 abends. The data's sign must match the signs allowed by the NUMPROC compiler option suboption that you specify.
- DATA(24) issues. Do not mix AMODE 24 programs with 31-bit data.

Note: In some cases, you can no longer set initial values of WORKING-STORAGE using the Language Environment STORAGE option. For details about storage changes, see “Language Environment option changes” on page 188.

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

- Locally: Each program separately
- 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 that you need, and then recompile, relink, rerun, and, if necessary, continue to debug.

7. Cut over to production mode.

When your testing shows that the entire application receives the expected results, you can move the entire unit over to production mode. (This assumes you have completed your migration to Language Environment.)

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

- Under z/OS, 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 that you are getting the results expected. After that, your source conversion task is completed.

Part 3. Upgrading programs

Chapter 5. Upgrading OS/VS COBOL source programs

There are differences between OS/VS COBOL language and Enterprise COBOL language that might require that you upgrade your programs.

This information will help you evaluate, from a language standpoint, which applications are good candidates for upgrading to Enterprise COBOL.

Besides the specific topics listed in this section, there has also been a change in tape user Label support. Support for the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and optionally the syntax: GO TO MORE-LABELS was removed in Enterprise COBOL V5

Also consider changes in reserved words as described in Appendix B, “COBOL reserved word comparison,” on page 233.

Enterprise COBOL provides COBOL 85 Standard support. When upgrading your OS/VS COBOL programs to Enterprise COBOL, you must convert them to COBOL 85 Standard programs in order to compile them with Enterprise COBOL.

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

- *VS COBOL for OS/VS Reference GC26-3857-04*
- *Enterprise COBOL Language Reference SC14-7381*

Tips:

1. *VS COBOL for OS/VS Reference* is no longer available from IBM.
2. There are special considerations related to CICS. OS/VS COBOL programs no longer run under CICS. Any OS/VS programs to be run under CICS must be upgraded to Enterprise COBOL.
3. 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 LANTLRVL(1) of OS/VS COBOL (Program 5740-CB1).
4. Information throughout this *Migration Guide* about OS/VS COBOL applies to OS/VS COBOL Release 2.4, with the latest service updates applied.

Comparing OS/VS COBOL to Enterprise COBOL

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

Language elements that require change (quick reference)

Table 8 on page 44 lists the language elements different in OS/VS COBOL and Enterprise COBOL. This table also lists conversion tools, if any, available to automate the conversion.

The language items listed below are described in detail throughout this section, 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

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL

Language element	Conversion tool	Page
Abbreviated combined relation conditions		"Abbreviated combined relation conditions and use of parentheses" on page 64
ACCEPT statement		"ACCEPT statement" on page 64
ALPHABETIC class changes	CCCA	"ALPHABETIC class changes" on page 72
ALPHABET clause changes—ALPHABET key word	CCCA	"ALPHABET-NAME clause changes: ALPHABET keyword " on page 73
Area A, periods in	CCCA	"Periods in Area A " on page 68
Arithmetic statement changes		"Arithmetic statement changes " on page 73
ASSIGN . . . OR	CCCA	"ASSIGN . . . OR" on page 58
ASSIGN TO <i>integer system-name</i>	CCCA	"ASSIGN . . . OR" on page 58
ASSIGN . . . FOR MULTIPLE REEL /UNIT	CCCA	"ASSIGN . . . FOR MULTIPLE REEL/UNIT " on page 58
ASSIGN clause changes— <i>assignment-name</i> forms	CCCA	"ASSIGN clause changes" on page 73
B symbol in PICTURE clause—changes in evaluation		"B symbol in PICTURE clause: changes in evaluation " on page 73
BDAM file handling	CCCA*	57

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
BLANK WHEN ZERO clause and asterisk (*) override		"BLANK WHEN ZERO clause and asterisk (*) override" on page 64
CALL identifier statement—B symbol in PICTURE clause		"B symbol in PICTURE clause: changes in evaluation " on page 73
CALL statement changes—procedure names and file names in USING phrase		"CALL statement changes " on page 74
CANCEL statement—B symbol in PICTURE clause		"B symbol in PICTURE clause: changes in evaluation " on page 73
CLOSE . . . FOR REMOVAL statement		"CLOSE . . . FOR REMOVAL statement" on page 65
CLOSE statement—WITH POSITIONING, DISP phrases	CCCA	"CLOSE statement: WITH POSITIONING, DISP phrases " on page 58
Combined abbreviated relation condition changes	CCCA	"Combined abbreviated relation condition changes" on page 74
Comparing group to numeric packed-decimal item		"Comparing group to numeric packed-decimal item" on page 65
COPY statement with associated names	CCCA	"COPY statement with associated names " on page 76
Communication feature		57

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
CURRENCY-SIGN clause changes—'/', '=', and 'L' characters		"CURRENCY-SIGN clause changes: '/', '=', and 'L' characters" on page 76
CURRENT-DATE special register	CCCA	"CURRENT-DATE special register " on page 58
DIVIDE . . . ON SIZE ERROR—change in intermediate results		"ON SIZE ERROR phrase: changes in intermediate results " on page 81
Dynamic CALL statements to programs with alternate entry points without an intervening CANCEL		"Dynamic CALL statements to ENTRY points " on page 76
EXAMINE statement	CCCA	"EXAMINE statement " on page 59
EXHIBIT statement	CCCA	"Corrective action for EXHIBIT NAMED" on page 59
EXIT PROGRAM/GOBACK statement changes		"EXIT PROGRAM/GOBACK statement changes " on page 76
FILE STATUS clause changes	CCCA	"FILE STATUS clause changes " on page 76
FILE-LIMIT clause of the FILE-CONTROL paragraph	CCCA	"FILE-LIMIT clause of the FILE-CONTROL paragraph " on page 60
Flow of control, no terminating statement		"Flow of control, no terminating statement" on page 65

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
FOR MULTIPLE REEL/UNIT	CCCA	"ASSIGN . . . FOR MULTIPLE REEL/UNIT " on page 58
GIVING phrase of USE AFTER STANDARD ERROR declarative	CCCA	"GIVING phrase of USE AFTER STANDARD ERROR declarative " on page 60
IF . . . OTHERWISE statement changes	CCCA	"IF . . . OTHERWISE statement changes " on page 79
Index names—nonunique		"Index names" on page 65
INSPECT statement—PROGRAM COLLATING SEQUENCE clause		"PROGRAM COLLATING SEQUENCE clause changes " on page 82
IS as an optional word		"Optional word IS " on page 81
ISAM file handling	CCCA	56
JUSTIFIED clause changes	CCCA	"JUSTIFIED clause changes " on page 79
LABEL RECORDS clause with TOTALING/TOTALED AREA	CCCA	"LABEL RECORDS clause with TOTALING/ TOTALED AREA phrases " on page 60
LABEL RECORD IS statement		"LABEL RECORD IS statement" on page 65
MOVE statement—binary value and DISPLAY value		"MOVE statement - binary value and DISPLAY value" on page 66
MOVE statements and comparisons—scaling changes		"MOVE statements and comparisons: scaling changes " on page 79

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
MOVE CORRESPONDING statement	CCCA	"MOVE CORRESPONDING statement" on page 66
MOVE statement—multiple TO specification		"MOVE statement - multiple TO specification" on page 67
MOVE ALL—TO PIC 99		"MOVE ALL - TO PIC 99" on page 67
MOVE statement—warning message for numeric truncation		"MOVE statement - warning message for numeric truncation" on page 67
MULTIPLY ... ON SIZE ERROR—change in intermediate results		"ON SIZE ERROR phrase: changes in intermediate results " on page 81
Nonunique program-ID names	CCCA	"PROGRAM-ID names, nonunique " on page 69
NOTE statement	CCCA	"NOTE statement " on page 60
Numeric class test on group items		"Numeric class test on group items" on page 80
Numeric data changes		"Numeric data changes" on page 80
Numeric-editing changes (PICTURE clause)		"PICTURE string " on page 69
OCCURS clause (order of phrases)		"OCCURS clause" on page 67
OCCURS DEPENDING ON—ASCENDING and DESCENDING KEY phrases		"OCCURS DEPENDING ON clause: ASCENDING and DESCENDING KEY phrase" on page 80

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
OCCURS DEPENDING ON—value for receiving items changed	CCCA	“OCCURS DEPENDING ON clause: value for receiving items changed ” on page 80
ON statement	CCCA	“ON statement ” on page 61
ON SIZE ERROR phrase—changes in intermediate results		“ON SIZE ERROR phrase: changes in intermediate results ” on page 81
OPEN statement failing for QSAM files (file status 39)		“OPEN statement failing for VSAM files (file status 39)” on page 61
OPEN statement failing for VSAM files (file status 39)		“OPEN statement failing for QSAM files (file status 39)” on page 61
OPEN statement with LEAVE, REREAD, and DISP phrases	CCCA	“OPEN statement with the LEAVE, REREAD, and DISP phrases ” on page 61
OPEN REVERSED statement		“OPEN REVERSED statement” on page 68
OTHERWISE clause changes		“IF . . . OTHERWISE statement changes ” on page 79
Paragraph names not allowed as parameters		“CALL statement changes ” on page 74
PERFORM statement—changes in the VARYING and AFTER phrases		“PERFORM statement: changes in the VARYING/ AFTER phrases ” on page 82

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
PERFORM statement—second UNTIL		"PERFORM statement - second UNTIL" on page 68
Periods, consecutive in any division		"Periods, consecutive in any division " on page 68
Periods in Area A	CCCA	"Periods in Area A " on page 68
Periods missing on paragraphs	CCCA	"Periods missing on paragraphs " on page 69
Periods missing at the end of SD, FD, or RD		"Periods missing at the end of SD, FD, or RD " on page 68
PICTURE clause (numeric-editing changes)		"PICTURE string " on page 69
PROGRAM COLLATING SEQUENCE clause changes		"PROGRAM COLLATING SEQUENCE clause changes " on page 82
Program-ID names, nonunique	CCCA	"PROGRAM-ID names, nonunique " on page 69
Qualification - using the same phrase repeatedly		"Qualification - using the same phrase repeatedly " on page 69
READ statement - redefined record keys in the KEY phrase		"READ statement - redefined record keys in the KEY phrase" on page 69
READ and RETURN statement changes—INTO phrase		"READ and RETURN statement changes: INTO phrase " on page 82

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
READY TRACE and RESET TRACE statements	CCCA	"READY TRACE and RESET TRACE statements " on page 61
RECORD CONTAINS n CHARACTERS clause		"RECORD CONTAINS n CHARACTERS clause " on page 69
RECORD KEY phrase and ALTERNATE RECORD KEY phrase		"RECORD KEY phrase and ALTERNATE RECORD KEY phrase" on page 69
REDEFINES clause in SD or FD entries	CCCA	"REDEFINES clause in SD or FD entries" on page 70
REDEFINES clause with tables		"REDEFINES clause with tables" on page 70
Relation conditions	CCCA	"Relation conditions" on page 70
REMARKS paragraph	CCCA	"REMARKS paragraph " on page 62
RENAMES clause—nonunique, nonqualified data names		"RENAMES clause - nonunique, nonqualified data names " on page 71
Report Writer statements	Report Writer Precompiler	55
RERUN clause changes		"RERUN clause changes " on page 82
RESERVE clause changes	CCCA	"RESERVE clause changes " on page 82
Reserved word list changes	CCCA	"Reserved word list changes" on page 83
SEARCH statement changes	CCCA	"SEARCH statement changes " on page 83

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
Segmentation changes—PERFORM statement in independent segments		"Segmentation changes: PERFORM statement in independent segments " on page 84
SELECT statement without a corresponding FD		"SELECT statement without a corresponding FD" on page 71
SELECT OPTIONAL clause changes	CCCA	"SELECT OPTIONAL clause changes " on page 84
SORT special registers		"SORT special registers " on page 84
SORT verb		"SORT verb" on page 71
SORT or MERGE		"SORT or MERGE" on page 71
Source language debugging changes		"Source language debugging changes " on page 84
START . . . USING KEY statement	CCCA	"START . . . USING KEY statement " on page 62
STRING statement—PROGRAM COLLATING SEQUENCE clause		"PROGRAM COLLATING SEQUENCE clause changes " on page 82
STRING statement—sending field identifier		"STRING statement - sending field identifier " on page 71
Subscripts out of range—flagged at compile-time		"Subscripts out of range flagged at compile time " on page 85
THEN as a statement connector	CCCA	"THEN as a statement connector " on page 63

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
TIME-OF-DAY special register	CCCA	"TIME-OF-DAY special register " on page 63
TOTALING/TOTALED AREA phrases in LABEL RECORDS clause	CCCA	"LABEL RECORDS clause with TOTALING/TOTALED AREA phrases " on page 60
TRANSFORM statement	CCCA	"TRANSFORM statement " on page 63
UNSTRING statement—PROGRAM COLLATING SEQUENCE clause		"PROGRAM COLLATING SEQUENCE clause changes " on page 82
UNSTRING statement—coding with 'OR', 'IS', or a numeric edited item	CCCA	"UNSTRING statement - coding with 'OR', 'IS', or a numeric edited item " on page 71
UNSTRING statement—multiple INTO phrases		"UNSTRING statement - multiple INTO phrases " on page 72
UNSTRING statements—subscript evaluation changes		"UNSTRING statements: subscript evaluation changes " on page 85
UPSI switches	CCCA	"UPSI switches " on page 86
USE AFTER STANDARD ERROR—GIVING phrase	CCCA	"GIVING phrase of USE AFTER STANDARD ERROR declarative " on page 60
USE BEFORE STANDARD LABEL statement	CCCA	"USE BEFORE STANDARD LABEL " on page 64

Table 8. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
VALUE clause—signed value in relation to the PICTURE clause	CCCA	“VALUE clause - signed value in relation to the PICTURE clause ” on page 72
VALUE clause—condition names	CCCA	“VALUE clause condition names ” on page 86
WHEN-COMPILED special register	CCCA	“WHEN-COMPILED special register ” on page 86
WRITE AFTER POSITIONING statement	CCCA	“WRITE AFTER POSITIONING statement” on page 87

* This is a partial conversion for handling BDAM files.

Converting to COBOL 85 Standard

To help you make the needed changes when upgrading to Enterprise COBOL, you can use any of several means, including the information provided elsewhere in this *Migration Guide*.

A brief description of two of the helpful mechanisms (CCCA and the MIGR option) follows. For additional information, see Appendix C, “Conversion tools for source programs,” on page 249.

Tip: Non-IBM tools are also available to help automate the conversion to COBOL 85 Standard.

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 Enterprise 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 253 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 Enterprise COBOL. 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 249 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*.

Language elements that require other products for support

Although some OS/VS COBOL language elements are not supported in Enterprise 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 Enterprise COBOL, you have the following considerations:

- Keep existing Report Writer code and use the Report Writer Precompiler
- Convert existing Report Writer code using the Report Writer Precompiler
- Run existing OS/VS COBOL-compiled Report Writer programs under Language Environment
- Report Writer language items affected

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 Enterprise COBOL compiler, your Report Writer applications can run above the 16-MB line. Through Enterprise COBOL, you can also extend their processing capabilities.

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

You can run Report Writer Precompiler as a separate precompiler, or incorporate it into the COBOL compilation by using the EXIT compiler option.

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 Enterprise 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 Enterprise COBOL but they cannot be mixed with Enterprise COBOL V5. If you want to mix Enterprise COBOL V5 programs with OS/VS COBOL Report Writer programs, you must convert all of the programs to use Enterprise COBOL V5, and use the Report Writer Precompiler.

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

Report Writer language items affected

The following Report Writer language items are accepted by Enterprise COBOL only when the Report Writer precompiler is installed:

- 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 249

Language elements that are not implemented

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

- ISAM file handling
- BDAM file handling
- Communication feature

With Enterprise 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 Enterprise COBOL.

The language elements affected and the conversion actions that 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

Enterprise COBOL does not support the processing of ISAM files, nor does z/OS V1.7 and later releases. You must convert ISAM files to VSAM/KSDS files before you move to z/OS V1.7 or later.

ISAM file handling language items affected

The following ISAM language items are not accepted by Enterprise COBOL:

- 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

Conversion options: Two conversion tools can help you convert ISAM files to VSAM/KSDS files. You can use either IDCAMS REPRO or CCCA. The IDCAMS REPRO facility will perform the conversion unless the file has a hardware

dependency. IDCAMS repro will only work for ISAM files on z/OS V1.6 or earlier. You must migrate ISAM to VSAM/KSDS before moving to z/OS V1.7 or later.

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 249.

BDAM file handling

Enterprise 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 following BDAM language items are not accepted by Enterprise COBOL:

- 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, however, you must provide the key algorithm. The CCCA conversion tool is described in Appendix C, “Conversion tools for source programs,” on page 249.

Communication feature

The Communication feature is not supported by Enterprise COBOL.

Communication language items affected

The following communication language items are not accepted by Enterprise COBOL:

- 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 runtime option of OS/VS COBOL is not supported. (The QUEUE runtime option is used only in an OS/VS COBOL program with a RECEIVE statement in a CD . . . FOR INITIAL INPUT.)

For more information, see the *IBM VS COBOL for OS/VS*, and the *IBM OS/VS COBOL Compiler and Library Programmer's Guide*.

Language elements that are not supported

Enterprise COBOL does not support the following OS/VS COBOL language elements. When upgrading to Enterprise 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 Enterprise 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 Enterprise 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. Enterprise 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 Enterprise 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)

Enterprise COBOL supports the DATE special register. It is valid only as the sending field in an ACCEPT statement. 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 one:

```
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/VIS COBOL accepted the EXAMINE statement; Enterprise COBOL does not.

Therefore, if your OS/VIS COBOL program contains coding similar to the following one:

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

Replace it in Enterprise 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/VIS COBOL accepted the EXHIBIT statement; Enterprise COBOL does not.

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

Corrective action for EXHIBIT NAMED

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

OS/VIS COBOL	Enterprise 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

Corrective action for 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 change displays the value of the specified data item only if the new value is different from the old:

OS/VIS COBOL	Enterprise COBOL
WORKING-STORAGE SECTION.	WORKING-STORAGE SECTION.
77 DAT-1 PIC X(8).	77 DAT-1 PIC X(8).

<pre> 77 DAT-2 PIC X(8). . . . EXHIBIT CHANGED DAT-1 DAT-2 </pre>	<pre> 77 DAT-2 PIC X(8). 77 DAT1-CMP PIC X(8). 77 DAT2-CMP PIC X(8). . . . IF DAT-1 NOT EQUAL TO DAT1-CMP DISPLAY DAT-1 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 </pre>
--	--

Corrective action for 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 change displays the value of the specified data item only if the new value is different from the old:

OS/VS COBOL	Enterprise COBOL
<pre> WORKING-STORAGE SECTION. 77 DAT-1 PIC X(8). 77 DAT-2 PIC X(8). . . . EXHIBIT CHANGED NAMED DAT-1 DAT-2 </pre>	<pre> WORKING-STORAGE SECTION. 77 DAT-1 PIC X(8). 77 DAT-2 PIC X(8). 77 DAT1-CMP PIC X(8). 77 DAT2-CMP PIC X(8). . . . IF DAT-1 NOT EQUAL TO DAT1-CMP DISPLAY "DAT-1 = " DAT-1 END-IF IF DAT-2 NOT EQUAL TO DAT2-CMP DISPLAY "DAT-2 = " DAT-2 END-IF MOVE DAT-1 TO DAT1-CMP MOVE DAT-2 TO DAT2-CMP </pre>

FILE-LIMIT clause of the FILE-CONTROL paragraph

OS/VS COBOL accepted the FILE-LIMIT clause and treats it as a comment; Enterprise 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. Enterprise 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.

Enterprise 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.

NOTE statement

OS/VS COBOL accepted the NOTE statement. Enterprise COBOL does not accept the NOTE statement. Therefore, for Enterprise COBOL delete all NOTE statements and use comment lines instead for the entire NOTE paragraph.

ON statement

OS/VS COBOL accepted the ON statement. Enterprise COBOL does not accept the ON statement.

The ON statement allows selective execution of statements it contains. Similar functions are provided in Enterprise 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 for a successful OPEN. In Enterprise COBOL, if the following conditions do not match, an OPEN statement in your program might not run 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 G, “Preventing file status 39 for QSAM files,” on page 283.

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 for a successful OPEN. In Enterprise COBOL they must match. The following rules apply to VSAM ESDS, KSDS, and RRDS file definitions:

Table 9. Rules for VSAM file definitions

File type	Rules
ESDS and KSDS VSAM	RECORDSIZE(<i>avg,m</i>) is specified where <i>avg</i> is the average size of the COBOL records, and is strictly less than <i>m</i> ; <i>m</i> is greater than or equal to the maximum size of a COBOL record.
RRDS VSAM	RECORDSIZE(<i>n,n</i>) is specified where <i>n</i> is greater than or equal to the maximum size of a COBOL record.

OPEN statement with the LEAVE, REREAD, and DISP phrases

OS/VS COBOL allowed the OPEN statement with the LEAVE, REREAD and DISP phrases. Enterprise 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 or use READ INTO.

READY TRACE and RESET TRACE statements

OS/VS COBOL allowed the READY TRACE and RESET TRACE statements. Enterprise COBOL does not support these statements.

To get function similar to the READY TRACE statement, you can use either Debug Tool, or the COBOL language available in the Enterprise COBOL compiler.

If you use Debug Tool, compile your program with the TEST option and use the following Debug Tool command:

```
"AT GLOBAL LABEL PERFORM;  
LIST LINES %LINE; GO; END-PERFORM;"
```

If you use the COBOL language, the Enterprise 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.

Enterprise COBOL does not accept the REMARKS paragraph. As a replacement, use comment lines beginning with an * in column 7 or use the floating comment indicator *>.

START . . . USING KEY statement

OS/VS COBOL allowed the START statement with the USING KEY phrase; Enterprise COBOL does not. In Enterprise 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
```

Enterprise COBOL does not support the use of THEN as a statement connector. Therefore, in Enterprise 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)

Enterprise 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 one:

```
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
```

TRANSFORM statement

OS/VS COBOL supported the TRANSFORM statement. Enterprise 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 Enterprise 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; Enterprise COBOL does not.

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

SEARCH ALL statements

If you have programs that contain SEARCH ALL statements and that were compiled with OS/VS COBOL, you may need to make some changes due to changes in the behavior of the SEARCH ALL statement

The new behavior for the SEARCH ALL statement is described in "Upgrading programs that have SEARCH ALL statements" on page 102.

Undocumented OS/VS COBOL extensions that are 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 Enterprise 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.

Enterprise COBOL supports most parenthesis usage as IBM extensions. However, there are two differences:

- Within the scope of an abbreviated combined relation condition, Enterprise COBOL does not support relational operators inside parentheses. For example:
$$A = B \text{ AND } (< C \text{ OR } D)$$
- Some incorrect usages of parentheses in relation conditions were accepted by OS/VS COBOL, but are not by Enterprise COBOL. For example:
$$(A = 0 \text{ AND } B) = 0$$

ACCEPT statement

OS/VS COBOL accepted the ACCEPT statement without the keyword FROM between the identifier and the mnemonic or function name.

Enterprise 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.

Enterprise COBOL does not accept these two language elements when they are specified for the same data description entry. Thus Enterprise 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 Enterprise 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. Enterprise 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'.
```

Enterprise COBOL does not allow such a comparison.

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 Enterprise 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.

Enterprise 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.

Enterprise COBOL does not accept such a LABEL RECORD clause.

MOVE statement - binary value and DISPLAY value

Although the Enterprise 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 COBOL coding that is not valid, 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	Enterprise 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 Enterprise 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 Enterprise COBOL when compiled with TRUNC(OPT).

MOVE CORRESPONDING statement

- OS/VS COBOL allowed more than one receiver with MOVE CORRESPONDING; Enterprise 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 Enterprise 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; Enterprise 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 Enterprise COBOL, change the data items in the receiver to have unique names.

MOVE statement - multiple TO specification

OS/VIS COBOL allowed the reserved word TO to precede each receiver in a MOVE statement. For example:

```
MOVE aa TO bb TO cc
```

In Enterprise COBOL, the above statement must be changed to:

```
MOVE aa TO bb cc
```

MOVE ALL - TO PIC 99

OS/VIS COBOL allowed group moves into a fixed numeric receiving field. For example:

```
MOVE ALL ' ' TO num1
```

where, num1 is PIC 99.

Enterprise COBOL does not allow the above case. In Enterprise COBOL, you can change the example to the following statement and it would be accepted:

```
MOVE ALL ' ' TO num1(1:)
```

MOVE statement - warning message for numeric truncation

OS/VIS 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.

```

You can get the same behavior with Enterprise COBOL if the compiler option DIAGTRUNC is in effect.

OCCURS clause

OS/VIS COBOL allowed a nonstandard order for phrases following the OCCURS clause; Enterprise COBOL does not.

For example, the following code sequence would be allowed in OS/VIS 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.

```

In Enterprise COBOL, the above example must be changed to the following code sequence:

```

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; Enterprise 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 UNTIL SSREJADV-EOF.

```

Enterprise COBOL does not allow a second UNTIL statement. It must be removed as shown in the following example:

```

PERFORM CHECK-FOR-MATCH THRU CHECK-FOR-MATCH-EXIT
    UNTIL PARM-COUNT = 7
    OR SSREJADV-EOF.

```

Periods in Area A

OS/VS COBOL allowed you to code a period in Area A following an Area-A item (or no item) that was not valid. With Enterprise 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.

Enterprise 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 code would be accepted by OS/VS COBOL, but would receive a severe (RC = 12) error and a warning (RC = 4) under Enterprise 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).

Enterprise 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) whereas Enterprise 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.
```

Enterprise COBOL does not accept statements such as the statements in the example above. 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. Enterprise 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; Enterprise 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.

Enterprise 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 Enterprise 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.

Enterprise COBOL does not allow this.

Record length, obtaining from QSAM RDW

In OS/VS COBOL, you can obtain the record length for files that have variable-length records from the RDW by using invalid negative subscripts.

In Enterprise COBOL, the RDW for variable files in the area preceding the record content is not available. To migrate from previous COBOL products, use the Format 3 RECORD clause in FD entries to set or obtain the length of variable records when the information is not in the record itself. The syntax contains RECORD IS VARYING DEPENDING ON *data-name-1*.

data-name-1 is defined in WORKING-STORAGE. After the compiler reads a variable record, the length of the data read is automatically stored at *data-name-1*. For example:

```
FILE SECTION.
FD THE-FILE RECORD IS VARYING DEPENDING ON REC-LENGTH.
01 THE-RECORD PICTURE X(5000) .
WORKING-STORAGE SECTION.
01 REC-LENGTH PICTURE 9(5) COMPUTATIONAL.
01 SAVED-RECORD PICTURE X(5000).
PROCEDURE DIVISION.
* Read a record of unknown length.
  READ THE-FILE.
  DISPLAY REC-LENGTH.
* or use REC-LENGTH to access the right amount of data:
  MOVE THE-RECORD (1:REC-LENGTH) TO SAVED-RECORD.
```

For more information about the RECORD clause, see the *Enterprise COBOL Language Reference*.

REDEFINES clause in SD or FD entries

Releases prior to OS/VS COBOL Release 2.4 accepted a REDEFINES clause in a level-01 SD or FD; Enterprise COBOL and OS/VS COBOL Release 2.4 do not.

For example, the following code sequence is not valid:

```
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 Enterprise 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.
```

Enterprise COBOL does not allow tables to be redefined, and issues a severe (RC = 12) message for the example above.

Relation conditions

Releases prior to OS/VS COBOL Release 2.4 accepted operators in relation conditions that are not valid. The following table lists the operators accepted by OS/VS COBOL Release 2.3 that are not accepted by Enterprise COBOL. It also shows the valid coding for Enterprise COBOL programs.

OS/VS COBOL R2.3	Enterprise 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/VSE COBOL program references a nonunique, nonqualified data name. However, Enterprise COBOL does not support the use of nonunique, nonqualified data names.

SELECT statement without a corresponding FD

OS/VSE COBOL accepted a SELECT statement that does not have a corresponding FD entry; Enterprise COBOL does not.

SORT verb

At early maintenance levels, the OS/VSE 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.
```

Enterprise COBOL does not accept statements such as the statements in the example above.

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/VSE COBOL accepted IS if used following ASCENDING KEY. Enterprise COBOL does not accept IS in this context. For example:

```
SORT SORT-FILE
  ASCENDING KEY IS SD-NAME-FIELD
  USING INPUT-FILE
  GIVING SORTED-FILE.
```

SORT or MERGE

With OS/VSE 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.

In Enterprise 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/VSE COBOL allowed a numeric sending field identifier that is not an integer. Under Enterprise COBOL, a numeric sending field identifier must be an integer.

UNSTRING statement - coding with 'OR', 'IS', or a numeric edited item

OS/VSE COBOL would not issue a diagnostic error message for UNSTRING statements containing any of the following instances of coding that is not valid:

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
```

Enterprise COBOL allows only nonnumeric data items as senders in the UNSTRING statement.

Enterprise 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
```

Enterprise 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 Enterprise COBOL, the VALUE clause literal must match the PICTURE clause and the sign must be removed.

Language elements that changed from OS/VS COBOL

Several OS/VS COBOL language elements are changed in Enterprise COBOL in order to conform to 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 because semantics changed.

For each element listed, there is a brief description pointing out the differences in results and what actions to take. Clarifying coding examples are also given as needed.

ALPHABETIC class changes

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

In Enterprise 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 Enterprise COBOL ALPHABETIC-UPPER class test for the OS/VS COBOL ALPHABETIC test.

ALPHABET-NAME clause changes: ALPHABET keyword

In OS/VS COBOL, the keyword ALPHABET was not allowed in the ALPHABET-NAMES clause.

In Enterprise COBOL, there is a keyword ALPHABET and it is required.

Arithmetic statement changes

Enterprise COBOL supports the following arithmetic items with enhanced accuracy:

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

Therefore, for arithmetic statements that contain these items, Enterprise 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

Enterprise 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 Enterprise COBOL.

B symbol in PICTURE clause: changes in evaluation

OS/VS COBOL accepted the PICTURE symbols A and B in definitions for alphabetic items.

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

This change can cause execution differences between OS/VS COBOL and Enterprise 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.

Enterprise COBOL CALL statements do not accept procedure names and accept only 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 and pass procedure names, you 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 Enterprise 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, this assembler routine in OS/VS COBOL is not valid in Enterprise COBOL :

```
CALL "ASMMOD" USING PARAMETER-1,
                     PARAGRAPH-1,
                     PARAGRAPH-2,

NEXT STATEMENT.
. . .
PARAGRAPH-1.
. . .
PARAGRAPH-2.
```

The sample code above should be rewritten as shown in the following example in order to compile with Enterprise 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 programmers code assembler programs that use 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 Enterprise 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 that 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.

Enterprise COBOL supports most parentheses usage as IBM extensions. However, there are some differences:

- Within the scope of an abbreviated combined relation condition, Enterprise 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 Enterprise 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

Enterprise COBOL does not accept this use of the optional word IS. In Enterprise COBOL, delete the word IS when used in this manner.

Enterprise 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 LANTLR(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...
```

Enterprise COBOL does not accept this COPY statement. To compile with Enterprise 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 LANTLR(1), accepted the '/' (slash) character, the 'L' character, and the '=' (equal) sign in the CURRENCY-SIGN clause.

Enterprise COBOL does not accept these characters as valid.

If these characters are present, you must remove them from the CURRENCY SIGN clause.

Dynamic CALL statements to ENTRY points

OS/VS COBOL allowed dynamic CALL statements to alternate entry points of subprograms without an intervening CANCEL, in some cases.

Enterprise 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 remained in its uncompleted state.

In Enterprise 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 Enterprise COBOL, status key values have been changed from those received from OS/VS COBOL:

- For QSAM files, see Table 10 on page 77.

- For VSAM files, see Table 11.

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 about Enterprise COBOL file status codes, see the *Enterprise COBOL Language Reference*.

Table 10. Status key values: QSAM files

OS/VS	Enterprise COBOL	Meaning
(undefined)	04	Wrong length record; successful completion
(undefined)	05	Optional file not available; 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 available
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

Table 11. Status key values: VSAM files

OS/VS	Enterprise COBOL	Meaning
(undefined)	14	On sequential READ for relative file, size of relative record number too large for relative key

Table 11. Status key values: VSAM files (continued)

OS/VS	Enterprise COBOL	Meaning
00	00	Successful completion
00	04	Wrong length record; successful completion
00	05	Optional file not available; successful completion
00	35	Nonoptional file not available. 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	Key not valid for a VSAM indexed or relative file; sequence error
22	22	Key not valid for a VSAM indexed or relative file; duplicate key and duplicates not allowed
23	23	Key not valid for a VSAM indexed or relative file; no record found
24	24	Key not valid for a VSAM indexed or relative file; attempt to write beyond file boundaries Enterprise 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 available
94	46	Sequential READ attempted with no valid next record
95	39	Conflict of fixed file attributes; OPEN fails
95	95	Not valid 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
```

Enterprise 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 LANTLRVL(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 Enterprise 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 Enterprise 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 LANTLRVL(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), where 'b' represents a blank.

With Enterprise 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 S99.  
  03 mm PIC S99.  
  03 dd PIC S99.
```

Enterprise COBOL issues an S-level message when the IF NUMERIC class test is used for GROUP items whose subordinates are signed.

Numeric data changes

Enterprise COBOL uses the NUMPROC compiler option to alter the code generated for decimal data. While NUMPROC(NOPFD) will cause processing more similar to OS/VS COBOL than NUMPROC(PFD), results are not the same in all cases. The results of MOVE statements, 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 contents of decimal data items that are not valid 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 and DESCENDING KEY phrase

OS/VS COBOL accepted a variable-length key in the ASCENDING and DESCENDING KEY phrases of the OCCURS DEPENDING ON clauses as an IBM extension.

In Enterprise COBOL, you cannot specify a variable-length key in the ASCENDING or DESCENDING KEY phrase.

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 Enterprise 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 Enterprise 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, Enterprise 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, Enterprise 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/VIS COBOL, the SIZE ERROR phrase for the DIVIDE and MULTIPLY statements applied to both intermediate and final results.

For Enterprise 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/VIS COBOL program depends upon SIZE ERROR detection for intermediate results, you might need to change it.

Optional word IS

For OS/VIS 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
```

Enterprise COBOL does not accept this use of the optional word IS. In Enterprise COBOL, delete the word IS when used in this manner.

Enterprise 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 Enterprise COBOL in such a PERFORM statement, the following results take 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 results:

```
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 Enterprise 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 Enterprise 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 Enterprise COBOL can select different records from under the FD or SD to use as the sending field. This only affects implicit elementary MOVE statements, 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; Enterprise 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
```

Enterprise 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 Enterprise COBOL. That is, the OS/VS COBOL phrase RESERVE 2 ALTERNATE AREAS is equivalent to RESERVE 3 AREAS in Enterprise COBOL.

Under OS/VS COBOL with LONGLVL(1), the interpretation of the RESERVE integer AREAS format differed from the interpretation of this format using Enterprise COBOL.

With LONGLVL(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 Enterprise COBOL.

Reserved word list changes

Differences exist between the reserved word list for Enterprise COBOL and OS/VS COBOL. Appendix B, "COBOL reserved word comparison," on page 233 contains a complete listing of reserved words.

SEARCH statement changes

In OS/VS COBOL, the ASCENDING and DESCENDING KEY data items could be specified either as the subject or as the object of the WHEN relation-condition of the SEARCH statement.

In Enterprise COBOL, the WHEN phrase data-name (the subject of the WHEN relation-condition) must be an ASCENDING or a DESCENDING KEY data item in this table element, and identifier-2 (the object of the WHEN relation-condition) must not be an ASCENDING or DESCENDING key data item for this table element.

OS/VS COBOL accepted the following statement; Enterprise COBOL does not:

```
WHEN VAL = KEY-1 ( INDEX-NAME-1 )
  DISPLAY "TABLE RECORDS OK".
```

The following SEARCH example will execute under both Enterprise 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.

In Enterprise 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 Enterprise 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: *Language Environment Installation and Customization*.

SORT special registers

The SORT-CORE-SIZE, SORT-FILE-SIZE, SORT-MESSAGE, and SORT-MODE-SIZE special registers are supported under Enterprise 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 Enterprise COBOL, the SORT-RETURN register only contains codes for successful (RC=0) and unsuccessful (RC=16) SORT completion.

Source language debugging changes

With Enterprise COBOL and OS/VS COBOL, you can debug source language with the USE FOR DEBUGGING declarative. Valid operands are shown in Table 12. Operands that are not valid in Enterprise COBOL must be removed from the OS/VS COBOL program. Use Debug Tool to achieve the same debugging results.

Table 12. USE FOR DEBUGGING declarative: valid operands

Debugging operands		Procedures are executed immediately:
OS/VS COBOL	Enterprise 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.

Table 12. USE FOR DEBUGGING declarative: valid operands (continued)

Debugging operands		Procedures are executed immediately:
OS/VS COBOL	Enterprise COBOL	
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 the <i>IBM VS COBOL for OS/VS</i> for a description.
ALL REFERENCES OF identifier-n	(none)	See the <i>IBM VS COBOL for OS/VS</i> for a description.
cd-name-1	(none)	See the <i>IBM VS COBOL for OS/VS</i> for a description.

Subscripts out of range flagged at compile time

Enterprise 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 Enterprise 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.
```

In OS/VS COBOL, subscript IND-1 would be reevaluated before the second receiver ZZ was filled.

In Enterprise COBOL, the subscript IND-1 is evaluated only once at the beginning of the execution of the UNSTRING statement.

In OS/VS COBOL with LONGLVL(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 about the behavior of this phrase in OS/VS COBOL, see the *IBM VS COBOL for OS/VS*.

In Enterprise 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/VSE COBOL with LANGLVL(1), will produce this result:

ID-R1	123	ID-D1	**	ID-C1	3
ID-R2	45678	ID-D2	**	ID-C2	5
ID-R3	90AB	ID-D3		ID-C3	4

OS/VSE COBOL with LANGLVL(2) and Enterprise COBOL will produce this result:

ID-R1	123	ID-D1	*	ID-C1	3
ID-R2	45678	ID-D2	*	ID-C2	5
ID-R3	90AB	ID-D3		ID-C3	4

UPSI switches

OS/VSE COBOL allowed references to UPSI switches and mnemonic names associated with UPSI. Enterprise COBOL allows condition-names only.

For example, if a condition-name is defined in the SPECIAL-NAMES paragraph, the following code examples have the same effect:

OS/VSE COBOL	Enterprise COBOL
SPECIAL-NAMES. UPSI-0 IS MNUPO	SPECIAL-NAMES. UPSI-0 IS MNUPO ON STATUS IS UPSI-0-ON OFF STATUS IS UPSI-0-OFF
PROCEDURE DIVISION IF UPSI-0 = 1 ... IF MNUPO = 0 ...	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/VSE 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.
```

Enterprise 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

Enterprise COBOL and OS/VSE 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/VSE COBOL the format is:

```
hh.mm.ssMMM DD, YYYY (hour.minute.secondMONTH DAY, YEAR)
```

In Enterprise 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; Enterprise COBOL does not.

In Enterprise COBOL, you can use the WRITE . . . AFTER ADVANCING statement to obtain behavior similar to WRITE . . . AFTER POSITIONING. The following two examples show OS/VS COBOL POSITIONING phrases and the equivalent Enterprise COBOL phrases.

When using WRITE . . . AFTER ADVANCING with literals:

OS/VS COBOL	Enterprise 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 nonliterals:

WRITE OUTPUT-REC AFTER POSITIONING SKIP-CC.

OS/VS COBOL		Enterprise 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

Restriction: With Enterprise 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 6. Compiling converted OS/VS COBOL programs

This section contains information about the following topics:

- Compiler options for converted programs
- Unsupported OS/VS COBOL compiler options
- Prolog format changes

Information specific to OS/VS COBOL or Enterprise COBOL is noted.

Compiler options for converted programs

Table 13 lists the compiler options that have special relevance to converted programs.

Table 13. 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 Enterprise 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 Enterprise COBOL BUFSIZE option.
DATA(24)	Use DATA(24) for Enterprise COBOL programs that are compiled with RENT and mixed with AMODE 24 assembler programs.
DIAGTRUNC	Use DIAGTRUNC to get numeric truncation flagging for MOVE statements. This is similar to the flagging in OS/VS COBOL.
NOSTGOPT	Use NOSTGOPT if you have non-referenced data items as eye-catchers or time/version stamps in WORKING-STORAGE. Use STGOPT only if you do not need unused data items.
NUMPROC	Use NUMPROC(NOPFD) plus the installation option 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. For other alternatives for sign representation, see the <i>Enterprise COBOL Programming Guide</i> .
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 <i>not</i> 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.

Table 13. Compiler options for converted OS/VS COBOL programs (continued)

Compiler option	Comments
TRUNC	<p>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 uses NOTRUNC as the default with OS/VS COBOL (except for select programs 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. For individual data items you can specify USAGE COMP-5 to get guaranteed nontruncation of binary data.</p> <p>High-order digits: Enterprise COBOL programs compiled with TRUNC(OPT) can give different results than OS/VS COBOL programs compiled with NOTRUNC. The main difference is that programs can lose nonzero high-order digits. For statements for which a loss of high-order digits might take place, Enterprise COBOL issues a diagnostic message indicating that you should ensure that at least one of the following conditions is met:</p> <ul style="list-style-type: none"> • The sending items will not contain large numbers. • The receiving items are defined with enough digits in the PICTURE clause to handle the largest sending data items.

Unsupported OS/VS COBOL compiler options

Table 14 shows the OS/VS COBOL compiler options that are not supported by Enterprise COBOL.

For a complete list of Enterprise COBOL compiler options, see Appendix E, “Option comparison,” on page 263.

Table 14. OS/VS COBOL compiler options not supported by Enterprise COBOL

OS/VS COBOL option	Enterprise COBOL equivalent
BATCH/NOBATCH	<p>Batch environment is always available (sequence of programs). CBL statements are always processed with Enterprise COBOL.</p> <p>Enterprise COBOL considerations for sequence of programs are described in the <i>Enterprise COBOL Programming Guide</i>.</p>
COUNT/NOCOUNT	Similar function is available in Debug Tool.
ENDJOB/NOENDJOB	ENDJOB behavior is always in effect.
LANGLVL(1/2)	The LANGLVL option is not available. Enterprise COBOL supports only COBOL 85 Standard.
LVL=A B C D/ NOLVL	FLAGSTD is used for FIPS flagging. ANSI COBOL 74 FIPS is not supported.
RES/NORES	The RES or NORES option is not available. With Enterprise COBOL, the object module is always treated 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.
STATE/NOSTATE	Function is available with the TEST option.
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.
SXREF/NOSXREF	The XREF option provides sorted SXREF output.

Table 14. OS/VS COBOL compiler options not supported by Enterprise COBOL (continued)

OS/VS COBOL option	Enterprise COBOL equivalent
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 Enterprise COBOL, the object deck runs only in the z/OS environment, not z/VM®. The OSDECK function is not supported.

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 Enterprise COBOL are Language Environment conforming, and thus have a different prolog format than with 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 Enterprise COBOL LIST compiler option to generate a listing that you can use to compare the OS/VS COBOL prolog format with the Enterprise COBOL prolog format.

Chapter 7. Upgrading VS COBOL II source programs

There are differences between the VS COBOL II language and the Enterprise COBOL language that might require that you modify your programs.

Your VS COBOL II programs will compile without change using the Enterprise COBOL compiler unless the programs meet one or more of the following conditions:

- Programs were compiled with the CMPR2 compiler option. Enterprise COBOL does not support the CMPR2/NOCMPR2 compiler option.
- Programs were compiled with VS COBOL II Release 3.x, and that contain one or more of three minor COBOL 85 Standard features that were subject to COBOL 85 Standard interpretation changes
- Programs were compiled with VS COBOL II Release 3.0 and that use ACCEPT . . FROM CONSOLE
- Programs use words which are now reserved in Enterprise COBOL
- Programs with undocumented VS COBOL II extensions
- Programs with SEARCH ALL statements
- Programs use the SIMVRD support
- Programs contain the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and optionally the syntax: GO TO MORE-LABELS. The support for these were removed in Enterprise COBOL Version 5.1

Upgrading VS COBOL II programs compiled with the CMPR2 compiler option

If your VS COBOL II source programs were compiled with the CMPR2 compiler option, you must convert them to NOCMPR2 programs in order to compile them with Enterprise COBOL. The CMPR2/NOCMPR2 compiler option is not supported in Enterprise COBOL. Enterprise COBOL programs behave as if NOCMPR2 was always in effect. For information about language differences between CMPR2 and NOCMPR2 (COBOL 85 Standard), see “Upgrading programs compiled with the CMPR2 compiler option” on page 107.

For information about tools that will help with the CMPR2 to NOCMPR2 conversion, see Appendix C, “Conversion tools for source programs,” on page 249.

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, IBM COBOL, and Enterprise 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 Release 3. Most likely you do not have these very minor differences in your programs because of their rarity. However, 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 are displayed in text that matches pseudo-text-1 of REPLACE statements.

Blank lines, which are interspersed in the matched text, will not be displayed in the output of the REPLACE statement. This change could affect the semantics of the resulting program since the line numbers could be different. (For example, if a program uses the USE FOR DEBUGGING declarative, the contents of DEBUG-ITEM might be different). If an Enterprise 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 Release 3.x.

Precedence of USE procedures

This difference affects the precedence of USE procedures relating to contained programs.

In VS COBOL II Release 3.x, a file-specific USE procedure always takes precedence over a mode-specific USE procedure. This precedence occurs 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 Enterprise COBOL, USE procedure precedence is based on a program by program level; from the current program to the containing program for that program, and so on to the outermost program.

The following message will be issued if an Enterprise COBOL generated program selects a different USE procedure than would have been used by the VS COBOL II Release 3.x program:

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 Release 3.x.

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 by using the actual length or the maximum length.

You might see a difference if the variable-length group meets all of the following criteria:

- 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. 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 NOCMR2 program (VS COBOL II Release 3.x, VS COBOL II Release 4, or Enterprise COBOL). They all use the actual length in this case.

MOVE ALL SPACES TO VAR-LEN-GROUP-B (1:) would give different results for the following programs compiled with NOCMR2:

- 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 Enterprise 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 and 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 might differ from VS COBOL II Release 3.x.

ACCEPT statement

One additional difference between later releases and VS COBOL II Release 3.0 involves the system input devices for the mnemonic-name suboption of the ACCEPT statement.

For VS COBOL II 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. For VS

COBOL II Release 3.1 through Release 4.0, an input record of 256 characters is assumed even if a logical record length of other than 80 characters is specified.

In Enterprise COBOL, the maximum logical record length allowed is 32,760 characters.

New reserved words

Enterprise COBOL has quite a few more reserved words than VS COBOL II. If your VS COBOL II programs use these reserved words as user-defined words, then they must be changed before you can compile your programs with Enterprise COBOL.

New reserved words

You can use CCCA to convert the reserved words automatically. For more information about the CCCA tool, see Appendix C, “Conversion tools for source programs,” on page 249.

CCCA is updated for reserved word conversions for Enterprise COBOL Version 5 by the PTF for APAR PM86253.

The following table shows the reserved words added to each subsequent release of COBOL. For a complete list of reserved words, see Appendix B, “COBOL reserved word comparison,” on page 233.

Table 15. New reserved words, by compiler.

Compiler	Reserved word
COBOL/370 V1R1	FUNCTION, PROCEDURE-POINTER
COBOL for MVS & VM V1R2	CLASS-ID, METAClass, RECURSIVE, END-INVOKE, METHOD, REPOSITORY, INHERITS, METHOD-ID, RETURNING, INVOKE, OBJECT, SELF, SUPER, LOCAL-STORAGE, OVERRIDE
COBOL for OS/390 & VM V2R1	Same as COBOL for MVS & VM
COBOL for OS/390 & VM V2R2	COMP-5, COMPUTATIONAL-5, EXEC, END-EXEC, SQL, TYPE, FACTORY
COBOL for OS/390 & VM V2R2 with PQ49375	EXECUTE
Enterprise COBOL V3R1	JNIENVPTR, NATIONAL, XML, END-XML, XML-EVENT, XML-CODE, XML-TEXT, XML-NTEXT, FUNCTION-POINTER
Enterprise COBOL V3R4	NATIONAL-EDITED, GROUP-USAGE
Enterprise COBOL V4R1	XML-NAMESPACE, XML-NAMESPACE-PREFIX, XML-NNAMESPACE, XML-NNAMESPACE-PREFIX
Enterprise COBOL V4R2	XML-SCHEMA Note: XML-INFORMATION is added as a reserved word with APAR PM85035.
Enterprise COBOL V5R1	XML-INFORMATION

Undocumented VS COBOL II extensions

The VS COBOL II compiler did not diagnose a period in Area A following an Area A item (or no item) that is not valid. In Enterprise COBOL, periods in Area A must be preceded by a valid Area A item.

SEARCH ALL statements

If you have programs that contain SEARCH ALL statements and that were compiled with VS COBOL II, you may need to make some changes due to changes in the behavior of the SEARCH ALL statement

The new behavior for the SEARCH ALL statement is described in “Upgrading programs that have SEARCH ALL statements” on page 102.

Upgrading programs that use SIMVRD support

This section describes the actions to upgrade programs that use SIMVRD support. Support for *COBOL simulated variable-length relative-record data sets (RRDS)* is removed for programs compiled with Enterprise COBOL Version 4 or later. These files must be changed to VSAM RRDS files.

In COBOL compilers that supported the NOCMR2 compiler option before Enterprise COBOL Version 4, it was possible to use *COBOL simulated variable-length RRDS* using a VSAM KSDS when you used the SIMVRD runtime option support.

The coding that you use in a COBOL program to identify and describe VSAM variable-length RRDS and COBOL simulated variable-length RRDS is similar. With Enterprise COBOL Version 4 you must use VSAM variable-length RRDS support. In general, the only action to migrate from COBOL simulated variable-length RRDS to VSAM variable-length RRDS support is to change the IDCAMS definition of the file.

Table 16. Steps for using variable-length RRDS

Step	VSAM variable-length RRDS	COBOL simulated variable-length RRDS
1	Define the file with the ORGANIZATION IS RELATIVE clause.	Same
2	Use FD entries to describe the records with variable-length sizes.	Same, but you must also code RECORD IS VARYING in the FD entry of every COBOL program that accesses the data set.
3	Use the NOSIMVRD runtime option.	Use the SIMVRD runtime option.
4	Define the VSAM file through access-method services as an RRDS.	Define the VSAM file through access-method services as follows: DEFINE CLUSTER INDEXED KEYS(4,0) RECORDSIZE(<i>avg</i> , <i>m</i>) <i>avg</i> Is the average size of the COBOL records; strictly less than <i>m</i> . <i>m</i> Is greater than or equal to the maximum size COBOL record + 4.

In step 2 for simulated variable-length RRDS, coding other language elements that implied a variable-length record format did not give you COBOL simulated variable-length RRDS. For example, the following elements alone did not cause the use of simulated variable-length RRDS access, and therefore did not require the SIMVRD runtime option:

- Multiple FD records of different lengths
- OCCURS . . . DEPENDING ON in the record definitions
- RECORD CONTAINS *integer-1* TO *integer-2* CHARACTERS

Use the REUSE IDCAMS parameter for files that contain records and that you will open for output.

- Define the file with the ORGANIZATION IS RELATIVE clause.
- Use FD entries to describe the records with variable-length sizes.
- Use the NOSIMVRD runtime option.
- Define the VSAM file through access-method services as an RRDS.

Errors: When you work with simulated variable-length relative data sets and true VSAM RRDS data sets, an OPEN file status 39 occurs if the COBOL file definition and the VSAM data-set attributes do not match.

For more reference information about the commands for using variable-length RRDS, see *z/OS DFSMS: Access Method Services for Catalogs*.

Chapter 8. Compiling VS COBOL II programs

This section contains information about the following topics:

- Compiler options for VS COBOL II programs
- Prolog format changes

Information specific to VS COBOL II or Enterprise COBOL is noted.

Compiler options for VS COBOL II programs

The Enterprise COBOL and VS COBOL II compilers are 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 from the ones you used with VS COBOL II, make sure you understand the possible effects on your applications. For information about converting your source programs from CMPR2 to NOCMPR2 see “Upgrading programs compiled with the CMPR2 compiler option” on page 107. For information about other compiler options, see the *Enterprise COBOL Programming Guide*.

Compiling with Enterprise COBOL

Table 17 lists the Enterprise COBOL compiler options that have special relevance to converted programs.

Table 17. Key Enterprise COBOL compiler options for VS COBOL II programs

Enterprise COBOL compiler options	Comments
PGMNAME	If compiling with Enterprise COBOL, use the PGMNAME(COMPAT) option to ensure that program names are processed in a manner compatible with VS COBOL II (and COBOL/370).
TEST	<p>The syntax of the TEST option is different in Enterprise COBOL than in VS COBOL II. The TEST option now has two suboptions. You can specify whether or not source file information is stored in the object and whether or not JUMPTO and GOTO commands are enabled for use with Debug Tool.</p> <p>TEST without any suboptions gives you TEST(NOJPD, SOURCE). For more information about the TEST option, see the <i>Enterprise COBOL Programming Guide</i>.</p>

Compiler options not supported in Enterprise COBOL

Table 18 on page 100 lists the VS COBOL II compiler options that are not supported in Enterprise COBOL. In some cases, the function of the VS COBOL II compiler option is mapped to an Enterprise COBOL compiler option, as described in the comments section.

Table 18. Compiler options not supported in Enterprise COBOL

VS COBOL II compiler options	Comments
CMPR2	The CMPR2 option is not supported. You must convert programs compiled with CMPR2 to COBOL 85 Standard in order to compile them with Enterprise COBOL.
FDUMP/NOFDUMP	<p>Enterprise COBOL does not provide the FDUMP compiler option. For existing applications, FDUMP is mapped to the Enterprise COBOL TEST compiler option, which can provide equivalent function and more.</p> <p>Language Environment generates a better formatted dump than VS COBOL II, regardless of the FDUMP option. The use of TEST enables Language Environment to include the symbolic dump of information about data items in the formatted dump.</p> <p>For information about how to obtain the Language Environment formatted dump at abnormal termination, see the <i>Language Environment Debugging Guide and Run-Time Messages</i>.</p> <p>If NOFDUMP is encountered, Enterprise COBOL issues a warning message because NOFDUMP is not supported.</p>
FLAGMIG	The FLAGMIG option is not supported in Enterprise COBOL. FLAGMIG requires CMPR2, which is not supported in Enterprise COBOL. To get similar migration flagging use CCCA, this <i>Migration Guide</i> , or a compiler released prior to Enterprise COBOL to compile programs that use FLAGMIG.
FLAGSAA	Enterprise COBOL does not support the FLAGSAA option. If FLAGSAA is specified, Enterprise COBOL issues a warning message.
NUMPROC(MIG)	Enterprise COBOL V5 does not support the NUMPROC(MIG) option. If NUMPROC(MIG) is specified, Enterprise COBOL V5 issues a warning message and the compilation will get the default setting for NUMPROC. This is either the user-customized default or the IBM default, which is NUMPROC(NOPFD).
RES/NORES	Enterprise COBOL does not provide the RES/NORES compiler option. If RES is encountered, Enterprise COBOL issues an informational message. If NORES is encountered, Enterprise COBOL issues a warning message.

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 Enterprise COBOL are Language Environment conforming, and thus have a different prolog format than in VS COBOL II. Existing applications that scan for date and time and user-level information need to be updated to the new format.

You can compile your programs with the Enterprise COBOL LIST compiler option to generate a listing that you can use to compare the VS COBOL II format with the Enterprise COBOL format.

Chapter 9. Upgrading IBM COBOL source programs

There are differences in COBOL language support between IBM COBOL and Enterprise COBOL.

This information will help you determine which IBM COBOL programs need source modifications in order to compile with Enterprise COBOL. For example, IBM COBOL programs compiled with the CMPR2 option require source modification because Enterprise COBOL does not support the CMPR2/NOCMPR2 compiler option.

This section contains information about the following items that you will need to consider when upgrading IBM COBOL source programs to Enterprise COBOL:

- Determining which programs require upgrade before you compile with Enterprise COBOL
- Upgrading SOM-based object-oriented (OO) COBOL programs
- SOM-based OO COBOL language elements that are not supported
- SOM-based OO COBOL language elements that are changed
- New reserved words in Enterprise COBOL
- Language Environment runtime considerations

For information about upgrading programs compiled with the CMPR2 compiler option, see “Migrating from the CMPR2 compiler option to NOCMPR2” on page 107.

For more information about migrating from the separate CICS translator to the integrated CICS translator, see “Migrating from the separate CICS translator to the integrated translator” on page 214

Determining which programs require upgrade before you compile with Enterprise COBOL

Many IBM COBOL programs will compile without change under Enterprise COBOL.

These programs, however, will need to be upgraded before compiling with Enterprise COBOL:

- Programs that have SEARCH ALL statements
- Programs that use the SIMVRD support
- Programs that use words which are now reserved in Enterprise COBOL
- Programs that have undocumented IBM COBOL extensions
- Programs that contain the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and optionally the syntax: GO TO MORE-LABELS. The support for these were removed in Enterprise COBOL Version 5.1
- Programs that use DATE FORMAT data types and/or DATEVAL, UNDATE or YEARWINDOW functions for Y2K
- Programs that have SOM-based object-oriented COBOL syntax
- Programs compiled with the CMPR2 compiler option

Upgrading programs that have SEARCH ALL statements

Enterprise COBOL has corrected errors in the implementation of the SEARCH ALL statement. SEARCH ALL statements in earlier releases of COBOL contained errors in the key comparison logic, which caused different results than might have been intended. In particular, the comparison did not produce the same result as an IF statement or a sequential SEARCH statement.

Length mismatch problem: a search argument is longer than the table key

The SEARCH ALL statement comparisons should pad an alphanumeric key with blanks or extend a numeric key with leading zeros if the key is shorter than the SEARCH argument. However, in V3R3 and earlier releases, SEARCH ALL ignored the excess characters in the argument in some cases. For example, an alphanumeric search argument of 01 ARG PIC X(6) containing "ABCDEF" would incorrectly match a table or array key of 05 MY-KEY PIC X(4) with value "ABCD". A search argument containing "ABCD??" (where ? is a blank) would match, as expected.

Similar problems occurred with a numeric search argument and keys. For example, a search argument of 01 ARG PIC 9(6) containing 123456 would incorrectly match a table or array key of 05 MY-KEY PIC 9(4) with value 3456. A search argument containing 003456 would match, as expected.

Sign mismatch problem: signed numeric argument and unsigned numeric key

A second problem occurs when the search argument is a signed numeric item and the table key is an unsigned numeric item. If the runtime value of the search argument is negative, such as -1234, programs compiled with V3R3 and earlier would match a table key of 1234. These comparisons should be done using the rules for a normal COBOL relation condition, and a negative argument such as -1234 should never match a table key that is unsigned.

The correction:

Enterprise COBOL corrected these problems. However, some applications compiled with earlier releases might depend on the incorrect behavior. You must identify and modify these applications before you move them to Enterprise COBOL Version 4 or later.

To assist you in identifying the programs and SEARCH ALL statements that are impacted by these corrections, the following compiler and runtime warning diagnostics are issued.

- Compiler messages: Enterprise COBOL compiler generates the following compiler diagnostics. Whether there is an actual impact depends on the contents of the argument at run time.
 - IGYPG3189-W for all SEARCH ALL statements that have a search argument that is longer than the table key, and hence might be impacted by the first problem
 - IGYPG3188-W when the search argument is a signed numeric item and the table key is an unsigned numeric item, and hence the program might be impacted by the second problem
- Runtime messages: The following runtime messages are generated. Programs that generate either of these runtime messages might be affected by the change.
 - IGZ0194W for all SEARCH ALL statements that have search arguments with excess bytes that are not blank or zero.

- IGZ0193W when the search argument is a signed numeric item with a negative value and the table key is an unsigned numeric item.

To migrate

To move an application to Enterprise COBOL Version 4 or later, do one of the following sets of steps:

- Act on the compiler messages:
 1. Compile your programs with Enterprise COBOL
 2. Review any SEARCH ALL statements that are flagged with compiler messages IGYPG3188-W or IGYPG3189-W; such statements are potentially impacted.

Tip: To minimize the possibility of incompatible results, you can force programmers at your site to correct these SEARCH ALL statements by changing the severity of these messages to E or S. To change the severity of these messages, you can use the MSGEXIT suboption of the EXIT compiler option. By doing this, the programs that get these messages cannot be run until the code is corrected. The sample user exit IGYMSGXT has sample code in it to change the severity of IGYPG3188-W and IGYPG3189-W, to IGYPG3188-S and IGYPG3189-S, respectively.

- Act on the runtime messages:
 1. Run the application in a test environment.
 2. Review any SEARCH ALL statements that generate runtime message IGZ0193W or IGZ0194W.

After you have identified which of the SEARCH ALL statements are affected, adjust the application logic appropriately by doing the following steps:

- For SEARCH ALL statements in which the search argument is longer than the table key, do one of the following actions:
 - Ensure that any bytes in the argument in excess of the key length are spaces or zeroes as appropriate.

Tip: When you have completed this investigation and if you decided not to change your programs, you can change the severity of IGYPG3188-W and IGYPG3189-W, to IGYPG3188-I and IGYPG3189-I, respectively, or suppress these messages entirely, by using the MSGEXIT suboption of the EXIT compiler options. This allows your programs to then compile with RC=0. The sample user exit IGYMSGXT has sample code in it to change the severity of IGYPG3188-W and IGYPG3189-W.

- Move the argument to a temporary data item of the same size as the key and use the temporary item as the search argument.
- Limit the range of the comparison with reference-modification. For example:
 - in the alphanumeric case of search argument 01 ARG PIC X(6) and key of 05 MY-KEY PIC X(4) use this:
WHEN MY-KEY (MY-INDEX) = ARG(1:4)
 - in the numeric case of search argument 01 ARG PIC 9(6) and array key of 05 MY-KEY PIC 9(4) use this:
WHEN MY-KEY (MY-INDEX) = ARG(3:4)

The second and third actions above will prevent the warning message in the future.

- For SEARCH ALL statements in which the search argument is signed and the table key is unsigned, ensure that the search argument is correctly initialized to a positive value before the SEARCH statement is run. Depending on the specific application logic in the COBOL program, it might be possible to make one of the following changes:
 - Change the data description of the argument to be unsigned.
 - Move the search argument to a temporary variable with no sign and use the temporary variable in the SEARCH ALL statement.

Either action will prevent the warning message in the future.

Upgrading programs that use SIMVRD support

This section describes the actions to upgrade programs that use SIMVRD support. Support for *COBOL simulated variable-length relative-record data sets (RRDS)* is removed for programs compiled with Enterprise COBOL Version 4 or later. These files must be changed to VSAM RRDS files.

In COBOL compilers that supported the NOCMPR2 compiler option before Enterprise COBOL Version 4, it was possible to use *COBOL simulated variable-length RRDS* using a VSAM KSDS when you used the SIMVRD runtime option support.

The coding that you use in a COBOL program to identify and describe VSAM variable-length RRDS and COBOL simulated variable-length RRDS is similar. With Enterprise COBOL Version 4 you must use VSAM variable-length RRDS support. In general, the only action to migrate from COBOL simulated variable-length RRDS to VSAM variable-length RRDS support is to change the IDCAMS definition of the file.

Table 19. Steps for using variable-length RRDS

Step	VSAM variable-length RRDS	COBOL simulated variable-length RRDS
1	Define the file with the ORGANIZATION IS RELATIVE clause.	Same
2	Use FD entries to describe the records with variable-length sizes.	Same, but you must also code RECORD IS VARYING in the FD entry of every COBOL program that accesses the data set.
3	Use the NOSIMVRD runtime option.	Use the SIMVRD runtime option.
4	Define the VSAM file through access-method services as an RRDS.	Define the VSAM file through access-method services as follows: <pre> DEFINE CLUSTER INDEXED KEYS(4,0) RECORDSIZE(<i>avg</i>,<i>m</i>) </pre> <p><i>avg</i> Is the average size of the COBOL records; strictly less than <i>m</i>.</p> <p><i>m</i> Is greater than or equal to the maximum size COBOL record + 4.</p>

In step 2 for simulated variable-length RRDS, coding other language elements that implied a variable-length record format did not give you COBOL simulated variable-length RRDS. For example, the following elements alone did not cause the use of simulated variable-length RRDS access, and therefore did not require the SIMVRD runtime option:

- Multiple FD records of different lengths
- OCCURS . . . DEPENDING ON in the record definitions
- RECORD CONTAINS *integer-1* TO *integer-2* CHARACTERS

Use the REUSE IDCAMS parameter for files that contain records and that you will open for output.

- Define the file with the ORGANIZATION IS RELATIVE clause.
- Use FD entries to describe the records with variable-length sizes.
- Use the NOSIMVRD runtime option.
- Define the VSAM file through access-method services as an RRDS.

Errors: When you work with simulated variable-length relative data sets and true VSAM RRDS data sets, an OPEN file status 39 occurs if the COBOL file definition and the VSAM data-set attributes do not match.

For more reference information about the commands for using variable-length RRDS, see *z/OS DFSMS: Access Method Services for Catalogs*.

Language Environment runtime considerations

Enterprise COBOL programs use the Language Environment STACK storage in several cases where IBM COBOL used HEAP storage. These cases include intrinsic functions UPPER-CASE and LOWER-CASE. Recompiling with Enterprise COBOL may result in a significant STACK storage usage difference. If the STACK is allocated below the 16-MB line and a large DSA (Dynamic Save Area) is needed, an insufficient storage error might occur.

To see the amount of storage that is required, compile your program with the compiler options MAP and LIST. Look for FuncResultTemp under the listing line:
***** AUTOMATIC MAP*****

You may need to reduce the amount of storage required or change to STACK=(...ANYWHERE..) to use storage above the line.

Numeric items with PICTURE P considerations

In Enterprise COBOL, when a data item whose PICTURE character-string contains the symbol P is referenced, the digit positions specified by the symbol P are considered to contain zeros when the sending operand is numeric.

For example, if you move a data item with PICTURE 9P VALUE IS 10 to data items with PICTURE 99 and PICTURE XX, it will result in the receiving fields containing 10 in both cases. But if a numeric item is not required, as when the item is compared to an alphanumeric item, the character value will be used. For example, an item with PICTURE 9P and VALUE IS 10 is equal to an item with PICTURE XX and VALUE IS "1 " (digit 1 followed by a space).

New reserved words in Enterprise COBOL

Enterprise COBOL has a few more reserved words than IBM COBOL. If your IBM COBOL programs use these reserved words as user-defined words, then they must be changed before you can compile your programs with Enterprise COBOL.

New reserved words

You can use CCCA to convert the reserved words automatically. For more information about the CCCA tool, see Appendix C, “Conversion tools for source programs,” on page 249.

CCCA is updated for reserved word conversions for Enterprise COBOL Version 5 by the PTF for APAR PM86253.

The following table shows the reserved words added to each subsequent release of COBOL. For a complete list of reserved words, see Appendix B, “COBOL reserved word comparison,” on page 233.

Table 20. New reserved words, by compiler.

Compiler	Reserved word
COBOL/370 V1R1	FUNCTION, PROCEDURE-POINTER
COBOL for MVS & VM V1R2	CLASS-ID, METAClass, RECURSIVE, END-INVOKE, METHOD, REPOSITORY, INHERITS, METHOD-ID, RETURNING, INVOKE, OBJECT, SELF, SUPER, LOCAL-STORAGE, OVERRIDE
COBOL for OS/390 & VM V2R1	Same as COBOL for MVS & VM
COBOL for OS/390 & VM V2R2	COMP-5, COMPUTATIONAL-5, EXEC, END-EXEC, SQL, TYPE, FACTORY
COBOL for OS/390 & VM V2R2 with PQ49375	EXECUTE
Enterprise COBOL V3R1	JNIENVPTR, NATIONAL, XML, END-XML, XML-EVENT, XML-CODE, XML-TEXT, XML-NTEXT, FUNCTION-POINTER
Enterprise COBOL V3R4	NATIONAL-EDITED, GROUP-USAGE
Enterprise COBOL V4R1	XML-NAMESPACE, XML-NAMESPACE-PREFIX, XML-NNAMESPACE, XML-NNAMESPACE-PREFIX
Enterprise COBOL V4R2	XML-SCHEMA Note: XML-INFORMATION is added as a reserved word with APAR PM85035.
Enterprise COBOL V5R1	XML-INFORMATION

SEARCH ALL statements

If you have programs that contain SEARCH ALL statements and that were compiled with IBM COBOL, you may need to make some changes due to changes in the behavior of the SEARCH ALL statement

You need to take some actions for certain programs that have SEARCH ALL statements and that were compiled with one of the following compilers:

- COBOL for OS/390 & VM
- COBOL for MVS & VM
- COBOL/370

The new behavior for the SEARCH ALL statement is described in “Upgrading programs that have SEARCH ALL statements” on page 102.

Migrating from the CMPR2 compiler option to NOCMPR2

If your COBOL programs were compiled with the CMPR2 option, you must convert them to NOCMPR2 programs to compile them with Enterprise COBOL. The CMPR2/NOCMPR2 option is not supported in Enterprise COBOL.

Enterprise COBOL programs behave as if NOCMPR2 is always in effect.

Upgrading programs compiled with the CMPR2 compiler option

Beginning with VS COBOL II Release 3.0, you could choose the COBOL 85 Standard behavior (without the Intrinsic Function module) by using NOCMPR2, or the COBOL 74 Standard behavior by using the CMPR2 compiler option. But with Enterprise COBOL, programs must be at the COBOL 85 Standard level.

The CMPR2 option provided the Standard COBOL 74 behavior as implemented by VS COBOL II Release 2, as well as *nonstandard* Release 2 extensions now implemented in COBOL 85 Standard. The NOCMPR2 option provided COBOL 85 Standard-conforming behavior and IBM extensions. This same mechanism was provided by IBM COBOL as an aid to allow delaying the upgrade from VS COBOL II Release 2 level code to COBOL 85 Standard level code. In Enterprise COBOL, this delay is not available.

Enterprise COBOL provides COBOL 85 Standard support whereas VS COBOL II Release 2, provided the COBOL 74 Standard support (with some COBOL 85 Standard features added in). The implementation of COBOL 85 Standard caused some language elements to behave in a manner that differs from the implementation of COBOL 74 Standard.

When referring to VS COBOL II Release 3 or later 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 3 or 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 3 or 4, with the NOCMPR2 compiler option
- IBM COBOL with the NOCMPR2 compiler option
- Enterprise COBOL

FLAGMIG

We use FLAGMIG to refer to the use of a pre-Enterprise COBOL compiler (VS COBOL II or IBM COBOL) that supports the CMPR2 and FLAGMIG options.

Tip: To aid you with migration to Enterprise COBOL, use a previous COBOL compiler that supports FLAGMIG and CMPR2 to flag the statements that need to be converted.

The language elements listed below are affected by the CMPR2/NOCMPR2 compiler option. The differences are explained in the sections that follow.

Table 21. Language elements different between CMPR2 and NOCMPR2

Language element	Page
ALPHABET clause of the SPECIAL-NAMES paragraph	"ALPHABET clause of the SPECIAL-NAMES paragraph" on page 109
ALPHABETIC class	"ALPHABETIC class" on page 109
CALL ... ON OVERFLOW	"CALL . . . ON OVERFLOW" on page 110
Comparisons between scaled integers and nonnumerics	"Comparisons between scaled integers and nonnumerics" on page 111
COPY...REPLACING statements using non-COBOL characters	"COPY ... REPLACING statements using non-COBOL characters" on page 112
COPY statement using national extension characters	"COPY statement using national extension characters" on page 114
File status codes	"File status codes" on page 114
Fixed filed attributes and DCB= parameters of JCL	"Fixed-file attributes and DCB= parameters of JCL" on page 116
Implicit EXIT PROGRAM	"Implicit EXIT PROGRAM" on page 117
OPEN statement failing for QSAM file (FILE STATUS 39)	"OPEN statement failing for QSAM files (FILE STATUS 39)" on page 118
OPEN statement failing for VSAM files (FILE STATUS 39)	"OPEN statement failing for VSAM files (FILE STATUS 39)" on page 119
PERFORM return mechanism	"PERFORM return mechanism" on page 119
PERFORM...VARYING...AFTER	"PERFORM ... VARYING ... AFTER" on page 121
PICTURE clause with "A"s and "B"s	"PICTURE clause with "A"s and "B"s" on page 123
PROGRAM COLLATING SEQUENCE	"PROGRAM COLLATING SEQUENCE" on page 125
READ INTO and RETURN INTO	"READ INTO and RETURN INTO" on page 126
RECORD CONTAINS n CHARACTERS	"RECORD CONTAINS n CHARACTERS" on page 128
SET...TO TRUE	"SET . . . TO TRUE" on page 128
SIZE ERROR on MULTIPLY and DIVIDE	"SIZE ERROR on MULTIPLY and DIVIDE" on page 130
UNSTRING operand evaluation	"UNSTRING operand evaluation" on page 132
UPSI switches	"UPSI switches" on page 138
Variable-length group moves	"Variable-length group moves" on page 138

ALPHABET clause of the SPECIAL-NAMES paragraph

Whether ALPHABET is a reserved word that must be specified in the ALPHABET clause depends on the setting of the CMPR2/NOCMPR2 option.

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 generates 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 for ALPHABET clause of the SPECIAL-NAMES paragraph::

Add the keyword ALPHABET to the ALPHABET clause.

ALPHABETIC class

Whether the ALPHABETIC class includes the 26 lowercase letters depends on the setting of the CMPR2/NOCMPR2 option.

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 generates 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 for the ALPHABETIC class:: 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

Whether the ON OVERFLOW condition is raised for errors other than "out of storage" errors depends on the setting of the CMPR2/NOCMPR2 option.

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. CMPR2 interpreted that definition to cover only the condition "not enough storage available to load the program."

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 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.

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 might 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 CMPR2, the ON OVERFLOW condition will be raised only for the "out of storage" errors, and the approach above is reasonable.

Running under NOCMPR2, the ON OVERFLOW condition might be raised for errors other than the "out of storage" errors, and therefore, the second call (inside the ON OVERFLOW phrase) might fail as well.

Corrective action for CALL . . . ON OVERFLOW:: 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 statements that you coded (in the ON OVERFLOW phrase) might not correct the error, and the subsequent CALL will fail as well.

In general, it is not possible for an Enterprise COBOL program to determine the actual cause of the error that raised the ON OVERFLOW condition.

Comparisons between scaled integers and nonnumerics

Comparisons between nonnumeric items and scaled numeric items are handled differently depending on the setting of the CMPR2/NOCMPR2 option.

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 for comparisons between scaled integers and nonnumerics: 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

Some non-COBOL characters in library text or COPY ... REPLACING statements are treated differently depending on the setting of the CMPR2/NOCMPR2 option.

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: COBOL 85 Standard disallows all non-COBOL characters and adds lowercase and the colon to the character set.

Lowercase alphabetic characters: "Lowercase" alphabetic characters, which were non-COBOL with CMPR2, are now in the set of legal COBOL characters with Enterprise COBOL. With CMPR2, 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, Enterprise 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:

CMPR2	NOCMPR2
After COPY & REPLACING	After COPY & REPLACING
1 XYZ PIC X.	1 XYZ PIC X.
1 ABC PIC XX.	1 XYZ PIC XX.

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.

Corrective action for lowercase alphabetic characters:: To obtain the same results when compiling CMPR2 programs under Enterprise COBOL, you must verify that all your REPLACING arguments are unique (even after folding to uppercase).

The colon (:) character: With CMPR2, the colon character was a non-COBOL character that COPY ... REPLACING allowed as part of its operands. This character is a legal COBOL separator under Enterprise COBOL.

```
COPY  A  REPLACING  == A ==      BY  == X ==  
                      == B ==      BY  == Y ==  
                      == A:B ==    BY  == Z ==.
```

If member A contains:

```
MOVE  A:B  TO  ID2.
```

These are the differences between CMPR2 and Enterprise COBOL after COPY ... REPLACING has been performed.

CMPR2	NOCMPR2
MOVE Z TO ID2.	MOVE X:Y TO ID2.

Because ":" is a separator under Enterprise COBOL, "A:B" is broken up into three separate tokens: "A" ":" and "B." The replacements for A and B are made first.

Message: This difference in behavior between the two releases is flagged by FLAGMIG.

IGYLI0160-W

****MIGR**** The colon will be treated as a separator under the "NOCMPR2" compiler option. Results may be different.

Corrective action for the colon (:) character:: To make the previous piece of code behave in the same manner as with CMPR2, change the REPLACING clauses to:

```
COPY  A  REPLACING  == A:B ==    BY  == Z ==  
                      == A ==      BY  == X ==  
                      == B ==      BY  == Y ==.
```

Characters that are not valid: Some 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 member above will be copied with the replacement executed. The Enterprise COBOL compiler will issue an E-level diagnostic message.

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 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.

Corrective action for characters that are not valid:: You should remove all non-COBOL characters from your source programs and COPY libraries, and replace them with COBOL characters.

This removal of non-COBOL characters will protect you against new problems in later releases of Enterprise COBOL. 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

Whether the characters @, #, and \$ can be coded in the text-name and library-name of the COPY statement depends on the setting of the CMPR2/NOCMPR2 option.

CMPR2: National extension characters @, #, and \$ are allowed in the text-name and library-name of the COPY statement. For example in COPY X\$3. the item will be copied.

NOCMPR2: The compiler will issue an E-level diagnostic message.

IGYLI0025-E

Name "X\$3" was invalid. It was processed as "X03".

Enterprise 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 Enterprise COBOL, code COPY "X\$3".

Message: The difference in behavior is 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.

Corrective action for the COPY statement that uses national extension characters:: You should change all national extension characters in your source programs and COPY libraries, to COBOL characters.

File status codes

The setting of the CMPR2/NOCMPR2 option affects which file status codes are returned and the amount of detail the codes provide about input-output operations.

CMPR2: File status codes based on the COBOL 74 Standard are returned with CMPR2.

NOCMPR2: The file status codes are enhanced with NOCMPR2. New and changed file status codes are returned, and more detail is provided about the

status of input-output 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 for file status codes:: Although there is no one-to-one mapping of the CMPR2 status codes to those in Enterprise COBOL, Table 22 shows, in general, the relationships between CMPR2 and NOCMPR2 file status codes. For a comprehensive definition of the Enterprise COBOL file status codes, see the *Enterprise COBOL Language Reference*.

Table 22. QSAM and VSAM file status codes with CMPR2 and NOCMPR2

VSAM file status codes		QSAM file status codes	
CMPR2	NOCMPR2	CMPR2	NOCMPR2
00	00	00	00
	04		04
	05		05
	14		07
	24		39
	35		44
	39		
	44		
02	02		
10	10	10	10
21	21		
22	22		
23	23		
24	24		
30	30	30	30
	39		39
		34	34
90	37	90	35
	90		37
			90
91	91		
92	38	92	38
	41		41
	42		42
	43		43
	44		46
	47		47
	48		48
	49		49
	92		92
93	93		
94	46		

Table 22. QSAM and VSAM file status codes with CMPR2 and NOCMPR2 (continued)

VSAM file status codes		QSAM file status codes	
CMPR2	NOCMPR2	CMPR2	NOCMPR2
95	39 95		
96	96		
97	97		

Fixed-file attributes and DCB= parameters of JCL

The handling of block sizes, record sizes, and other fixed-file attributes is different between CMPR2 and NOCMPR2. You might need to change your programs and your JCL to migrate to NOCMPR2.

CMPR2: In CMPR2 programs, fixed-file attribute checking is only done at READ/WRITE time, if done at all. An OPEN statement could succeed even if some fixed-file attributes were inconsistent. For example, an OPEN could succeed with different record sizes in:

- RECORD CONTAINS x clause
- JCL DCB=(LRECL=y)
- Actual data-set label

NOCMPR2: In NOCMPR2 programs, COBOL 85 Standard requires that fixed-file attribute checking be done in many cases. As a result, a program with inconsistent fixed file attributes might fail at OPEN time rather than have problems later. The OPEN could fail with either runtime message IGZ0035S or file status 39 (if a file status clause is specified). See Appendix G, "Preventing file status 39 for QSAM files," on page 283 for more information about preventing file status 39 for QSAM files.

A common source of fixed file attribute inconsistency problems is the DCB= parameter of the JCL DD statement for your file.

Messages: There are no **MIGR** messages for these differences, because fixed-file attributes can be specified outside of the source program.

Recommendation for DCB= parameters of JCL:

It is strongly recommended that you take advantage of features of DFSMS and COBOL that let the system determine the block size. (In general, you should not specify DCB= attributes except in the few cases mentioned in the *Enterprise COBOL Programming Guide*.)

These are the recommendations:

- For new files, let z/OS determine the block size. To take advantage of system-determined block size:
 - Code BLOCK CONTAINS 0 in your source program or use the BLOCK0 compiler option.
 - Do not code RECORD CONTAINS 0 in your source program.
 - Do not code a BLKSIZE value in the JCL DD statement.
- For existing blocked data sets, use the existing file block size:
 - Code BLOCK CONTAINS 0 in your source program or use the BLOCK0 compiler option.

- Do not code a BLKSIZE value in the ddname definition.

The one case where you might consider putting BLKSIZE in the JCL is if you require a specific block size for a new file and you need the flexibility to modify that block size without recompiling your program. In this case, follow these guidelines:

- Code BLOCK CONTAINS 0 in your source program or use the BLOCK0 compiler option.
- Code a BLKSIZE value in the ddname definition (DCB=(BLKSIZE=xxx) in the JCL DD statement).

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 statements above, a compiler warning diagnostic message is issued to suggest that you should analyze the program 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 runtime 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 be issued:

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 for Implicit EXIT PROGRAM:: 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.

OPEN statement failing for QSAM files (FILE STATUS 39)

There is a difference in the way CMPR2 and NOCMPR2 handle fixed-file attributes for QSAM files for OPEN statements.

CMPR2: The fixed file attributes for QSAM files do not need to match between COBOL program file definition, JCL, or data-set label for a successful file OPEN.

NOCMPR2: If the following items are inconsistent, an OPEN statement in your program might not run successfully:

- The fixed file attributes of a file from the data set label
- The fixed file attributes specified in the JCL DD statement for a file
- The attributes specified for that file in the SELECT and FD statements of your COBOL program

Inconsistencies 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.

Message: There are no ****MIGR**** messages for this difference, because fixed-file attributes can be specified outside of the source program.

Corrective action for OPEN statement failing for QSAM files (FILE STATUS 39): To prevent common file status 39 problems, see Appendix G, “Preventing file status 39 for QSAM files,” on page 283.

OPEN statement failing for VSAM files (FILE STATUS 39)

There is a difference in the way CMPR2 and NOCMPR2 handle RECORDSIZE defined in VSAM files associated with IDCAMS.

In CMPR2, the RECORDSIZE defined in your VSAM files associated with IDCAMS was not required to match your COBOL program file definition for successful file OPEN.

CMPR2: The RECORDSIZE defined in your VSAM files associated with IDCAMS was not required to match your COBOL program file definitions for successful file OPEN.

NOCMPR2: The RECORDSIZE defined in your VSAM files associated with IDCAMS are required to match the file definitions for those files in your COBOL program for successful file OPEN.

Message: There are no **MIGR** messages for this difference, because the VSAM RECORDSIZE attribute is outside of the source program.

Corrective action for OPEN statement failing for VSAM files (FILE STATUS 39): Change your program file definitions or the RECORDSIZE defined in your VSAM files associated with IDCAMS to match according to the following table. The following rules apply to VSAM ESDS, KSDS, and RRDS file definitions:

Table 23. Rules for VSAM file definitions

File type	Rules
ESDS and KSDS VSAM	RECORDSIZE(<i>avg</i> , <i>m</i>) is specified where <i>avg</i> is the average size of the COBOL records, and is strictly less than <i>m</i> ; <i>m</i> is greater than or equal to the maximum size of a COBOL record.
RRDS VSAM	RECORDSIZE(<i>n</i> , <i>n</i>) is specified where <i>n</i> is greater than or equal to the maximum size of a COBOL record.

PERFORM return mechanism

There is a difference in the way CMPR2 and NOCMPR2 handle out-of-line PERFORM statements that might require corrective action.

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.

CMR2: 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 might not be intended. The problem might occur whenever all of the following conditions 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 might 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 NOCMR2, 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 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 for the PERFORM return mechanism:: 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

Certain identifiers in the VARYING phrase of the PERFORM statement are set and incremented differently depending on whether CMPR2 or NOCMPR2 is in effect.

Identifiers are set and increment 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 an earlier COBOL compiler 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

Only PERFORM ... VARYING with the AFTER phrase is affected.

For example, compiling the program under an earlier COBOL compiler with the CMPR2 and FLAGMIG compiler options causes 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 for PERFORM . . . VARYING . . . AFTER: 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 example above 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 conversion above 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

A data item that has the symbol B in its PICTURE clause is treated either as alphabetic or alphabetic-edited depending on whether CMPR2 or NOCMPR2 is in effect.

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 CMPR2 to Enterprise COBOL, relating to the INITIALIZE, STRING, CALL and CANCEL verbs.

Message: If a program is compiled with the CMPR2 and FLAGMIG options, a message is issued 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.

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 incompatibility 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 be issued 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".
```

Corrective action for the INITIALIZE verb: To initialize any of these reclassified data items in the same manner as they had been previously, change the original statement for the first example above to the following statement:

```
INITIALIZE ALPHA REPLACING  
    ALPHANUMERIC-EDITED DATA BY ALL "3".
```

In the second example, which shows a group of possibly mixed types, INITIALIZE should be supplemented with an additional phrase. For example:

```
INITIALIZE GROUP1 REPLACING  
    ALPHABETIC DATA BY ALL "5"  
    ALPHANUMERIC-EDITED DATA BY ALL "5".
```

Important: Adding this extra phrase could cause conflicts if you 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 CMPR2 programs have an alphabetic item defined with the symbol B in this position of the STRING verb, these statements will get a severe error message from Enterprise COBOL 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.

Corrective action for the STRING verb: Because a STRING statement with CMPR2 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 with CMPR2.

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 CMPR2 programs with alphabetic items defined with the symbol B will get a severe error message. For example, the following program would have worked with CMPR2, 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.

To compile with Enterprise COBOL, change the definition of CALLDN to all alphabetic or alphanumeric or add a new data-name that redefines 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

The truth value of nonnumeric comparisons determined by the PROGRAM COLLATING SEQUENCE clause might be different under CMPR2 and NOCOMPR2.

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. 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 might 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 must be rewritten to avoid this dependency on the CMPR2 behavior.

READ INTO and RETURN INTO

READ (or RETURN) with the INTO phrase might be performed differently for CMPR2 and NOCMPR2 for fixed-length files that have multiple 01-level record descriptions in which at least one of the descriptions is numeric or numeric-edited.

When deciding which record description to use as the sending field for an implicit MOVE statement, 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 might 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 be issued:

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 might be selected for the sending field for the move under the "NOCMPR2" compiler option.

Corrective action for the READ INTO and RETURN INTO phrases:: 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", under both 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 statements:

```
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.

After you have detected an incompatibility, 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 required 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.

Its behavior is different under CMPR2 and NOCMPR2.

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.

A program that has this difference might get a file status 39 on OPEN after compiling with Enterprise COBOL.

Corrective action for the RECORD CONTAINS n CHARACTERS clause:: To maintain current behavior, remove the RECORD CONTAINS clauses. This change results in FILE1 and FILE2 both 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.

SET . . . TO TRUE

SET ... TO TRUE has different effects depending on whether CMPR2 or NOCMPR2 is in effect.

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 change 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.

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")
```

Corrective action for the JUSTIFIED clause: 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 required 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 required 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

SIZE ERROR behaves differently depending on whether CMPR2 or NOCMPR2 is in effect.

The COBOL 74 Standard and the COBOL 85 Standard 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 *Enterprise COBOL Programming Guide* describes the use and definition of intermediate results. One such definition says that an intermediate result will have at most 30-digits (31-digits with ARITH(EXTEND)).

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 (or 31-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 (or 31-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 (or 31-digit) (intermediate) result.

```
COMPUTE C D = A * B ON SIZE ERROR...
```

This behavior 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 (or 31-digit) (intermediate) result. Consequently, the MULTIPLY and COMPUTE statements become equivalent in this regard. This behavior 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 can occur due to precision of intermediate results exceeding 30-digits.

If, at run time, truncation actually does occur, the following message will be issued:

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 for the SIZE ERROR on MULTIPLY and DIVIDE:: The CMPR2 behavior of affected programs cannot be preserved without extensive and complex recoding. Such programs must be rewritten to avoid this dependency on the CMPR2 behavior.

UNSTRING operand evaluation

Subscripting, indexing, and length calculation associated with the UNSTRING statement might generate different results depending on whether CMPR2 or NOCMPR2 is in effect.

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 change can lead to different results when certain dependencies exist between id-2 through id-9.

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.

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 Standard COBOL 74 and COBOL 85 Standard and are not addressed here. Generally, results will be unpredictable.

Compiling the program with the CMPR2 and FLAGMIG options causes the compiler to issue messages for all UNSTRING statements that might contain such dependencies.

Any UNSTRING statements not flagged with one of these messages will generate identical results under CMPR2 and NOCMR2.

All UNSTRING statements flagged with message 2222 will require changes to guarantee identical results.

Corrective action for the UNSTRING OPERAND evaluation:: 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 situation 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-OD00BJ PIC 9 VALUE IS 5.
02 VLG-1-GR.
03 VLG-1-ODO PIC X OCCURS 1 TO 9 TIMES
    DEPENDING ON VLG-1-OD00BJ.
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 "NOCMR2" compiler option.

Enterprise COBOL 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 example:

```
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 depend on one of the INTO receivers.

For example:

```
01 DEL
02 OCC-DEL-1 PIC X OCCURS 9 TIMES.
02 VLEN-DEL-2-OD00BJ 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-OD00BJ.

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-OD00BJ,
        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.

For an UNSTRING statement to be affected by this change, the flagged INTO identifier must depend 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-OD00BJ PIC 9 VALUE IS 9.
02 ODO-R-3.
03 FILLER PIC X OCCURS 1 TO 9 TIMES
    DEPENDING ON R-3-OD00BJ.

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-OD00BJ,
        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 that are 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 rules:

- 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-OD00BJ,
    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, or refers to a variably located item.

For an UNSTRING statement to be affected by this change, the flagged DELIMITER IN identifier must depend on one of the receivers in a preceding INTO phrase.

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-OD00BJ PIC 9 VALUE IS 9.
02 OD0-D-3.
03 FILLER PIC X OCCURS 1 TO 9 TIMES
    DEPENDING ON D-3-OD00BJ.

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-OD00BJ,
        R-3 DELIMITER IN OD0-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 that are 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-OD00BJ,
        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.

For an UNSTRING statement to be affected by this change, the flagged COUNT IN identifier must depend on one of the receivers in a preceding INTO phrase.

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, the values that are 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 example above 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

Condition-names for the UPSI switches must be defined and referenced differently depending on whether CMPR2 or NOCMPR2 is in effect.

CMPR2: UPSI switches can 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 statements above 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 for UPSI switches:: 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

The calculation of the length of a sending or receiving ODO object can vary depending on whether CMPR2 or NOCMPR2 is in effect.

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 the 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 behavior 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 when 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 generates 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 affected verbs are:

- 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 for variable-length group moves:: You can take the following steps:

- See if any of your COBOL programs have the variable-length data movement statements by compiling them with the CMPR2 and FLAGMIG compiler options. This completion will flag all variable-length group moves with receivers that contain their own ODO objects and are not complex ODO items.
- 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.
- 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).`

Upgrading SOM-based object-oriented (OO) COBOL programs

SOM-based object-oriented COBOL applications are not supported with Enterprise COBOL. OO COBOL syntax has been retargeted for Java-based object-oriented programming to facilitate interoperation of COBOL and Java.

The Java-based OO COBOL is not compatible with SOM-based OO COBOL, and is not intended as a migration path for OO COBOL programs. In most cases you should rewrite your OO COBOL into procedural COBOL in order to use the Enterprise COBOL compiler. It is possible that you could use the new OO COBOL syntax in place of your existing SOM-based OO syntax, but it is not a straightforward conversion.

For more information about the considerations that apply when you upgrade your IBM COBOL programs that contain SOM-based OO COBOL statements to Enterprise COBOL, see “SOM-based OO COBOL language elements that are not supported” and “SOM-based OO COBOL language elements that are changed” on page 141.

SOM-based OO COBOL language elements that are not supported

When you migrate COBOL applications that use SOM-based OO programming to the Java-based OO programming in Enterprise COBOL, be aware of the following SOM elements that are not supported.

Calls to SOM

Calls to SOM services are not supported.

INHERITS clause

- Specification of more than one class name on the INHERITS clause of the CLASS-ID paragraph (multiple inheritance) is not supported.
- COBOL classes must be ultimately derived from the `java.lang.Object` class (rather than `SOMObject` or `SOMClass`). Specification of `SOMObject` as a base class in the INHERITS clause is not supported.
- Specification of `SOMClass` as a base class in the INHERITS clause (defining metaclasses) is not supported. Java-based OO COBOL classes can specify a FACTORY section, defining static methods that are logically part of the class-object for the class.

INVOKE

- Argument lists on INVOKE statements and parameter lists for methods are restricted to data types that map to Java types and that are passed BY VALUE.

- Specification of a class-name that qualifies SUPER in the INVOKE statement is not supported. For example you cannot use:

```
INVOKE C OF SUPER "foo"
```

However, the following syntax continues to be supported in Enterprise COBOL:

```
INVOKE SUPER "foo"
```

METACLASS clauses

- The METACLASS IS clause of the CLASS-ID paragraph is not supported.
- The METACLASS OF clause from the USAGE clause, which defines object references, is not supported.

METHODS

- The OVERRIDE clause of the METHOD-ID paragraph is not supported.
- Use of methods from SOM base classes such as somNew, somFree, and somInit are not supported.

Compiler options IDLGEN and TYPECHK

The IDLGEN and TYPECHK options are not available. Both compiler options require SOM-based OO COBOL, which is not available with Enterprise COBOL.

SOM-based OO COBOL language elements that are changed

When you migrate applications that use SOM-based OO programming to the Java-based OO programming in Enterprise COBOL, be aware of the following elements that are changed in Enterprise COBOL.

External names

- External class names that are defined in the REPOSITORY paragraph must be defined with Java naming conventions for fully qualified class names, rather than the CORBA rules of formation for class names.
- Method names that are specified as literals use Java naming conventions rather than CORBA naming conventions.

INVOKE

Instead of somNew, object instances are created with the syntax:

```
INVOKE classname NEW ...
```

METHODS

COBOL methods can override inherited methods and can be overloaded, according to Java rules. However, the OVERRIDE clause is not required or supported on the METHOD-ID paragraph in these cases.

OBJECTS

- Instead of somNew, object instances are created with the syntax:

```
INVOKE classname NEW ...
```
- Object instances are freed through Java automatic garbage collection, rather than somFree.
- Object instance data is initialized through VALUE clauses or user-written initialization methods, rather than with somInit.
- OBJECT and END OBJECT syntax must be specified unless the class does not specify any object instance data or object instance methods.

Chapter 10. Compiling IBM COBOL programs

This section contains information about the following topics:

- Default compiler option changes from IBM COBOL
- Compiler options for IBM COBOL programs
- Compiler options not available in Enterprise COBOL

Information specific to IBM COBOL or Enterprise COBOL is noted.

Default compiler options for IBM COBOL programs

The Enterprise COBOL compiler has slightly different default compiler options than IBM COBOL. The compiler options DBCS, FLAG(I,I), RENT, and XREF(FULL) are now default values in the product configuration that is shipped from IBM. The default values for IBM COBOL were NODBCS, FLAG(I), NORENT, and NOXREF.

The DBCS option might cause problems for CICS programs if you are using the COBOL2 CICS translator option. The fix is to use the COBOL3 translator option.

Compiler options for IBM COBOL programs

The Enterprise COBOL and IBM COBOL compilers are very similar. If you will be using the same compiler options that were used in your current IBM COBOL applications, some internal changes might take effect, but basically the behavior is unchanged.

If you do change compiler options settings from the settings you used with IBM COBOL applications, make sure you understand the possible effects on your applications. For information about other compiler options, see the *Enterprise COBOL Programming Guide*.

There are some new compiler options in Enterprise COBOL compared to compiler options in IBM COBOL. Table 24 lists the options that affect compatibility between IBM COBOL and Enterprise COBOL.

Table 24. Compiler options for IBM COBOL programs

Compiler option	Comments
ARITH	Use ARITH(COMPAT) to get the same results as COBOL/370, Release 1, thru COBOL for OS/390 & VM, Version 2 Release 1 for intermediate results in arithmetic statements.

Table 24. Compiler options for IBM COBOL programs (continued)

Compiler option	Comments
INTDATE	<p>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.</p> <p>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.</p> <p>If you choose INTDATE(LILIAN) as your installation default, you should recompile all of your COBOL/370, Release 1 programs (and any IBM COBOL programs that used INTDATE(ANSI)) that use intrinsic functions to ensure that all of your code uses the Lilian integer date standard. This method is the safest, because you can store integer dates and pass them between programs, even between PL/I, COBOL, and C programs, and know that the date processing will be consistent.</p>
PGMNAME	<p>Use PGMNAME(COMPAT) to ensure that program names are processed in a manner similar to COBOL/370, Release 1.</p>
NSYMBOL	<p>Controls the interpretation of the "N" symbol used on literals and PICTURE clauses, indicating whether national or DBCS processing is assumed.</p> <p>NSYMBOL(DBCS) provides compatibility with previous releases of IBM COBOL and VS COBOL II.</p>
TRUNC	<p>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 Enterprise COBOL and 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).</p> <p>For example, in a program compiled with TRUNC(BIN), a data item declared like this</p> <pre>01 X pic 9(2) binary.</pre> <p>correctly supported binary values from 0 through only 32767 in prior releases, but with Version 2 Release 2 now supports values of 0 through 65535.</p> <p>This support necessarily yields different arithmetic results than were obtained with the prior releases, if these very large unsigned binary values were inadvertently used.</p>

Compiler options not available in Enterprise COBOL

Most compiler options that are available in IBM COBOL can be used when you compile with Enterprise COBOL except for the following compiler options:

Table 25. Compiler options not available in Enterprise COBOL

Compiler option	Comments
ANALYZE	The ANALYZE option is not available with Enterprise COBOL. Use the CICS, SQL, and ADATA options instead.
CMPR2	The CMPR2 option is not available. You must convert programs compiled with CMPR2 to COBOL 85 Standard to compile them with Enterprise COBOL
EVENTS	The EVENTS option is not available. To emulate the COBOL/370 EVENTS compiler option: <ol style="list-style-type: none">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.
FLAGMIG	The FLAGMIG option is not available. FLAGMIG requires CMPR2, which is not available with Enterprise COBOL. Use CCCA, this <i>Migration Guide</i> , or a compiler released prior to Enterprise COBOL to compile programs using FLAGMIG.
IDLGEN	The IDLGEN option is not available. IDLGEN requires SOM-based OO COBOL, which is not available with Enterprise COBOL.
NUMPROC(MIG)	Enterprise COBOL does not support the NUMPROC(MIG) option in versions after Version 4. If NUMPROC(MIG) is specified, Enterprise COBOL issues a warning message and the compilation will get the default setting for NUMPROC. This is either the user-customized default or the IBM default, which is NUMPROC(NOPFD).
TYPECHK	The TYPECHK option is not available. TYPECHK requires SOM-based OO COBOL, which is not available with Enterprise COBOL.
WORD(NOOO)	<p>If you have existing IBM COBOL programs that were compiled with the WORD(NOOO) compiler option, they must be changed if they use any of the following reserved words: CLASS-ID, END-VOKE, INHERITS, INVOKE, LOCAL-STORAGE, METAClass, METHOD, METHOD-ID, OBJECT, OVERRIDE, RECURSIVE, REPOSITORY, RETURNING, SELF, SUPER.</p> <p>The IGYCNOOO reserved word table is not shipped with the Enterprise COBOL compiler.</p>

Chapter 11. Upgrading programs from Enterprise COBOL Version 3

To compile with Enterprise COBOL Version 5, programs that use any of several features must be upgraded.

Programs that contain any of the following language features need to be modified:

- Programs with Search ALL
- Programs that use XML PARSE
- Programs that use XML GENERATE
- Programs that use new reserved words as user words. For details, see “New reserved words” on page 96.
- Programs that use SIMVRD feature
- Label declaratives. Programs that contain the format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and optionally the syntax: GO TO MORE-LABELS. The support for these were removed in Enterprise COBOL Version 5
- Programs using DATE FORMAT and windowed date functions. For details, see “Changes in millenium language extensions in IBM Enterprise COBOL for z/OS, Version 5” on page 174.

Consider the following topics:

SEARCH ALL statements

If you have programs that contain SEARCH ALL statements and that were compiled with Enterprise COBOL releases V3R1 through V3R4 before the installation of the PTF for APAR PK16765.

These actions are required due to changes in the behavior of the SEARCH ALL statement with the PTF for APAR PK16765 .

Tip: You can tell if your compiler has this PTF installed by looking at the page header in the compiler listing. The modification level was changed by this PTF from 0 to 1. If the product name in the header looks like this: "Enterprise COBOL for z/OS 3.4.1", your compiler has the PTF installed.

Upgrading programs that have SEARCH ALL statements

Enterprise COBOL has corrected errors in the implementation of the SEARCH ALL statement. SEARCH ALL statements in earlier releases of COBOL contained errors in the key comparison logic, which caused different results than might have been intended. In particular, the comparison did not produce the same result as an IF statement or a sequential SEARCH statement.

Length mismatch problem: a search argument is longer than the table key

The SEARCH ALL statement comparisons should pad an alphanumeric key with blanks or extend a numeric key with leading zeros if the key is shorter than the SEARCH argument. However, in V3R3 and earlier releases, SEARCH ALL ignored the excess characters in the argument in some cases. For example, an alphanumeric

search argument of 01 ARG PIC X(6) containing "ABCDEF" would incorrectly match a table or array key of 05 MY-KEY PIC X(4) with value "ABCD". A search argument containing "ABCD??" (where ? is a blank) would match, as expected.

Similar problems occurred with a numeric search argument and keys. For example, a search argument of 01 ARG PIC 9(6) containing 123456 would incorrectly match a table or array key of 05 MY-KEY PIC 9(4) with value 3456. A search argument containing 003456 would match, as expected.

Sign mismatch problem: signed numeric argument and unsigned numeric key

A second problem occurs when the search argument is a signed numeric item and the table key is an unsigned numeric item. If the runtime value of the search argument is negative, such as -1234, programs compiled with V3R3 and earlier would match a table key of 1234. These comparisons should be done using the rules for a normal COBOL relation condition, and a negative argument such as -1234 should never match a table key that is unsigned.

The correction:

Enterprise COBOL corrected these problems. However, some applications compiled with earlier releases might depend on the incorrect behavior. You must identify and modify these applications before you move them to Enterprise COBOL Version 4 or later.

To assist you in identifying the programs and SEARCH ALL statements that are impacted by these corrections, the following compiler and runtime warning diagnostics are issued.

- Compiler messages: Enterprise COBOL compiler generates the following compiler diagnostics. Whether there is an actual impact depends on the contents of the argument at run time.
 - IGYPG3189-W for all SEARCH ALL statements that have a search argument that is longer than the table key, and hence might be impacted by the first problem
 - IGYPG3188-W when the search argument is a signed numeric item and the table key is an unsigned numeric item, and hence the program might be impacted by the second problem
- Runtime messages: The following runtime messages are generated. Programs that generate either of these runtime messages might be affected by the change.
 - IGZ0194W for all SEARCH ALL statements that have search arguments with excess bytes that are not blank or zero.
 - IGZ0193W when the search argument is a signed numeric item with a negative value and the table key is an unsigned numeric item.

To migrate

To move an application to Enterprise COBOL Version 4 or later, do one of the following sets of steps:

- Act on the compiler messages:
 1. Compile your programs with Enterprise COBOL
 2. Review any SEARCH ALL statements that are flagged with compiler messages IGYPG3188-W or IGYPG3189-W; such statements are potentially impacted.

Tip: To minimize the possibility of incompatible results, you can force programmers at your site to correct these SEARCH ALL statements by changing the severity of these messages to E or S. To change the severity of these messages, you can use the MSGEXIT suboption of the EXIT compiler option. By doing this, the programs that get these messages cannot be run until the code is corrected. The sample user exit IGYMSGXT has sample code in it to change the severity of IGYPG3188-W and IGYPG3189-W, to IGYPG3188-S and IGYPG3189-S, respectively.

- Act on the runtime messages:
 1. Run the application in a test environment.
 2. Review any SEARCH ALL statements that generate runtime message IGZ0193W or IGZ0194W.

After you have identified which of the SEARCH ALL statements are affected, adjust the application logic appropriately by doing the following steps:

- For SEARCH ALL statements in which the search argument is longer than the table key, do one of the following actions:
 - Ensure that any bytes in the argument in excess of the key length are spaces or zeroes as appropriate.

Tip: When you have completed this investigation and if you decided not to change your programs, you can change the severity of IGYPG3188-W and IGYPG3189-W, to IGYPG3188-I and IGYPG3189-I, respectively, or suppress these messages entirely, by using the MSGEXIT suboption of the EXIT compiler options. This allows your programs to then compile with RC=0. The sample user exit IGYMSGXT has sample code in it to change the severity of IGYPG3188-W and IGYPG3189-W.

- Move the argument to a temporary data item of the same size as the key and use the temporary item as the search argument.
- Limit the range of the comparison with reference-modification. For example:
 - in the alphanumeric case of search argument 01 ARG PIC X(6) and key of 05 MY-KEY PIC X(4) use this:
WHEN MY-KEY (MY-INDEX) = ARG(1:4)
 - in the numeric case of search argument 01 ARG PIC 9(6) and array key of 05 MY-KEY PIC 9(4) use this:
WHEN MY-KEY (MY-INDEX) = ARG(3:4)

The second and third actions above will prevent the warning message in the future.

- For SEARCH ALL statements in which the search argument is signed and the table key is unsigned, ensure that the search argument is correctly initialized to a positive value before the SEARCH statement is run. Depending on the specific application logic in the COBOL program, it might be possible to make one of the following changes:
 - Change the data description of the argument to be unsigned.
 - Move the search argument to a temporary variable with no sign and use the temporary variable in the SEARCH ALL statement.

Either action will prevent the warning message in the future.

Upgrading Enterprise COBOL Version 3 programs that have XML PARSE statements

This topic provides solutions for upgrading Enterprise COBOL Version 3 programs that have XML PARSE statements.

Enterprise COBOL V5 does not support the same XML parser as Enterprise COBOL V3. In particular, the z/OS System Services XML parser is now the parser used by XML PARSE in Enterprise COBOL V5 while Enterprise COBOL V3 used an XML parser that was part of the COBOL runtime library.

The z/OS System Services XML parser provides the following benefits:

- The latest IBM parsing technology
- Offload of COBOL XML parsing to zAAP specialty processors
- Improved support for parsing XML documents that use XML namespaces
- Direct support for parsing XML documents encoded in UTF-8 Unicode
- Support for parsing very large XML documents, a buffer at a time

To migrate your Enterprise COBOL Version 3 programs that have XML PARSE statements to Enterprise COBOL Version 5, change the programs to reflect the new, changed, and discontinued XML parsing events. For details, see “Migrating from the old XML parser to the new XML parser.”

Migrating from the old XML parser to the new XML parser

New, changed, unchanged, and discontinued events with XML PARSE in Enterprise COBOL V5:

- You can migrate your programs to use the z/OS System Services XML parser (referred to as the *new parser*) from the XML parser that was part of the COBOL runtime environment (referred to as the *old parser*) after you understand the differences between the new and the old parsers. Some of these differences are described in terms of new, changed, unchanged, and discontinued events when the new parser is in effect:
 - **ATTRIBUTE-CHARACTER event (discontinued)**
 - **New parser:** The ATTRIBUTE-CHARACTER event no longer occurs. All entity references, including predefined ones, are now included in the ATTRIBUTE-CHARACTERS event, unless there is an unresolved entity reference, in which case an EXCEPTION event is signaled.
 - **Old parser:** The ATTRIBUTE-CHARACTER event occurs for predefined entity references only. The five predefined entity references are shown in Table 26 on page 157. XML-TEXT or XML-NTEXT contains the single character that corresponds with the predefined entity reference in the attribute value. Character references are signaled as ATTRIBUTE-NATIONAL-CHARACTER events.
 - **To migrate to the new parser:** Remove references to the ATTRIBUTE-CHARACTER event and integrate any actions for this event into your ATTRIBUTE-CHARACTERS event handling.
 - **ATTRIBUTE-CHARACTERS event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT could have a substring of the value for the ATTRIBUTE-CHARACTERS event. XML-TEXT or XML-NTEXT could also contain a complete string of the value even if the value contains a character reference or an entity reference.

- **Old parser:** XML-TEXT or XML-NTEXT has only a substring of the value for the ATTRIBUTE-CHARACTERS event when the value contains a character reference or an entity reference.
- **To migrate to the new parser:** You might have to modify your code that handles the ATTRIBUTE-CHARACTERS event to handle more than one event even if your attribute values do not contain character or entity references. You might also have to change your code to process ATTRIBUTE-CHARACTERS as a single event where your code was handling ATTRIBUTE-CHARACTERS as multiple events.
- **ATTRIBUTE-NAME event (changed)**
 - **New parser:** For attribute names that are not in a namespace, XML-TEXT or XML-NTEXT contains the attribute name, and the namespace special registers are all empty and have length zero. Attributes with names in a namespace are always prefixed and have the form:
`prefix:local-part = AttValue`

XML-TEXT or XML-NTEXT contains the local-part, XML-NAMESPACE or XML-NNAMESPACE contains the namespace and XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix.
 - **Old parser:** For all attribute names, XML-TEXT or XML-NTEXT contains the complete attribute name, even if the name is prefixed (indicating that the name belongs to a namespace).
 - **To migrate to the new parser:** Either change your code to process the separate parts of the namespace, or change your code to reconstruct the complete attribute name from the separate parts in XML-TEXT, XML-NAMESPACE-PREFIX, and XML-NAMESPACE, or XML-NTEXT, XML-NNAMESPACE-PREFIX, and XML-NNAMESPACE.
- **ATTRIBUTE-NATIONAL-CHARACTER event (changed)**
 - **New parser:** Character references that can be represented in the EBCDIC encoding of the XML document are resolved and included in the ATTRIBUTE-CHARACTERS event.

Unrepresentable character references are expressed as ATTRIBUTE-NATIONAL-CHARACTER events, as for COMPAT.
 - **Old parser:** Regardless of the type of the XML document specified by identifier-1 in the XML PARSE statement, XML-TEXT is empty and XML-NTEXT contains the single national character corresponding with the (numeric) character reference.
 - **To migrate to the new parser:** Possibly no change will be required, but be aware that with the old parser, the national character might have an EBCDIC equivalent, whereas with the new parser, the national character is known to have no representation in the EBCDIC encoding of the document.
- **COMMENT event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT could have a substring of the value for the COMMENT event.
 - **Old parser:** XML-TEXT or XML-NTEXT always has the complete string of the value for the COMMENT event.
 - **To migrate to the new parser:** You might have to modify your code that handles the COMMENT event to handle more than one event if you get a substring of the COMMENT value in XML-TEXT or XML-NTEXT. If that is the case, you get two or more COMMENT events in succession and you

would concatenate strings together to re-create the complete string of the value. You cannot distinguish a comment that is split in this way from a sequence of distinct comments.

– **CONTENT-CHARACTER event (discontinued)**

- **New parser:** The CONTENT-CHARACTER event no longer occurs. All entity references, including predefined ones, are now included in the CONTENT-CHARACTERS event unless there is an unresolved entity reference, in which case an UNRESOLVED-REFERENCE event or an EXCEPTION event is signaled.
- **Old parser:** The CONTENT-CHARACTER event occurs for predefined entity references only. The five predefined entity references are shown in Table 26 on page 157. XML-TEXT or XML-NTEXT contains the single character that corresponds with the predefined entity reference in the attribute value. Character references are signaled as CONTENT-NATIONAL-CHARACTER events.
- **To migrate to the new parser:** Remove references to the CONTENT-CHARACTER event and integrate any actions for this event into your CONTENT-CHARACTERS event handling.

– **CONTENT-CHARACTERS event (changed)**

- **New parser:** XML-TEXT or XML-NTEXT could have a substring of the content for the CONTENT-CHARACTERS event. XML-TEXT or XML-NTEXT could also contain a complete string of the content even if the content contains a character reference or an entity reference.
- **Old parser:** XML-TEXT or XML-NTEXT has only a substring of the content for the CONTENT-CHARACTERS event when the content contains a character reference or an entity reference.
- **To migrate to the new parser:** You might have to modify your code that handles the CONTENT-CHARACTERS event to handle more than one event even if your attribute values do not contain character or entity references. You might also have to change your code to process CONTENT-CHARACTERS as a single event where your code was handling CONTENT-CHARACTERS as multiple events.

– **CONTENT-NATIONAL-CHARACTER event (changed)**

- **New parser:** Character references that can be represented in the EBCDIC encoding of the XML document are resolved and included in the CONTENT-CHARACTERS event.
Unrepresentable character references are expressed as CONTENT-NATIONAL-CHARACTER events, as for COMPAT.
- **Old parser:** Regardless of the type of the XML document specified by identifier-1 in the XML PARSE statement, XML-TEXT is empty, and XML-NTEXT contains the single national character corresponding with the (numeric) character reference.
- **To migrate to the new parser:** Possibly no change will be required, but be aware that with the old parser, the national character might have an EBCDIC equivalent, whereas with the new parser, the national character is known to have no representation in the EBCDIC encoding of the document.

– **DOCUMENT-TYPE-DECLARATION event (changed)**

- **New parser:** XML-TEXT or XML-NTEXT contains the name of the root element, as specified in the document type declaration. The parser processes entity declarations and default attribute values in the internal DTD subset, and ignores the rest of the text in the document type declaration.

- **Old parser:** XML-TEXT or XML-NTEXT contains the entire document type declaration.
- **To migrate to the new parser:** If having the whole document type declaration is important, you might have to modify your code that handles the DOCUMENT-TYPE-DECLARATION event to acquire the information directly from your XML document.
- **ENCODING-DECLARATION event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the encoding name. The encoding declaration is not used by the parser, so you might get incorrect characters passed through that would cause the parser to signal an EXCEPTION event from which you can't recover.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the encoding name. If there are errors in the encoding of the document, you would get an EXCEPTION event from which you might be able to recover and continue.
 - **To migrate to the new parser:** Check your document before parsing or specify your encoding using the CODEPAGE compiler option or by using the WITH ENCODING phrase on the XML PARSE statement.
- **END-OF-CDATA-SECTION event (changed)**
 - **New parser:** All XML special registers except XML-EVENT, XML-CODE and XML-INFORMATION are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT always contains the string "]]>".
 - **To migrate to the new parser:** If the string "]]>" is acquired from the END-OF-CDATA-SECTION event, change your code to manually return it using a literal, or data item initialized with the value "]]>".
- **END-OF-DOCUMENT event (no change)**
 - The 2 parsers have the same behavior for the END-OF-DOCUMENT event.
 - **To migrate to the new parser:** No change required.
- **END-OF-ELEMENT event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the local part of the end element tag or empty element tag name. If the element name is in a namespace, XML-NAMESPACE or XML-NNAMESPACE contains the namespace, otherwise these special registers are empty with length zero. If the element name is in a namespace and is prefixed (of the form "prefix:local-part"), XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix, otherwise these special registers are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the complete element tag name, including any prefix. If the element name is not in a namespace, there is no difference between the old and new parsers for END-OF-ELEMENT.
 - **To migrate to the new parser:** If the element name is not in a namespace, then no change is required. If the element name is in a namespace, change your code to not use the complete element name, or reconstruct the complete element name from the separate parts in the XML text and namespace special registers.
- **END-OF-INPUT event (new)**
 - **New parser:** The END-OF-INPUT event indicates the end of a segment of an XML document.
 - **Old parser:** The END-OF-INPUT event does not occur.
 - **To migrate to the new parser:** With the old parser, your document is in one segment, so no change is required to change to the new parser.

- **EXCEPTION event (changed)**
 - **New parser:** XML-CODE contains the unique return code and reason code identifying the exception. See the following section "Other differences" for a description of XML-CODE differences. XML-TEXT or XML-NTEXT contains the document fragment up to the point of the error or anomaly that caused the EXCEPTION event. All other XML special registers except XML-EVENT and XML-INFORMATION are empty with length zero. It is not possible to continue from any EXCEPTION event.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entire document that has been parsed up to the point of the EXCEPTION event. It is possible to continue from some EXCEPTION events.
 - **To migrate to the new parser:** You might have to change your code or documents if they depend on being able to recover from EXCEPTION events.
- **NAMESPACE-DECLARATION event (new)**
 - **New parser:** XML-TEXT and XML-NTEXT are both empty with length zero. XML-NAMESPACE or XML-NNAMESPACE contains the declared namespace. If the namespace is "undeclared" by specifying the empty string, XML-NAMESPACE and XML-NNAMESPACE are empty with length zero. XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix if the attribute name for the namespace declaration is of the form "xmlns:prefix", otherwise, if the declaration is for the default namespace and the attribute name is "xmlns", XML-NAMESPACE-PREFIX and XML-NNAMESPACE-PREFIX are both empty with length zero.
 - **Old parser:** The NAMESPACE-DECLARATION event does not occur.
 - **To migrate to the new parser:** If you get the NAMESPACE-DECLARATION event after migrating to the new parser, see the descriptions in this table of ATTRIBUTE-NAME, END-OF-ELEMENT and START-OF-ELEMENT event changes.
- **PROCESSING-INSTRUCTION-DATA event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT could have a substring of the value for the PROCESSING-INSTRUCTION-DATA event.
 - **Old parser:** XML-TEXT or XML-NTEXT always has the complete string of the value for the PROCESSING-INSTRUCTION-DATA event.
 - **To migrate to the new parser:** You might have to modify your code that handles the PROCESSING-INSTRUCTION-DATA event to handle more than one event if you get a substring of the PROCESSING-INSTRUCTION-DATA value in XML-TEXT or XML-NTEXT. If that is the case, you get two or more PROCESSING-INSTRUCTION-DATA events, each one preceded by its matching PROCESSING-INSTRUCTION-TARGET event. You would then concatenate the PROCESSING-INSTRUCTION-DATA substrings together to reconstitute the complete data string.
- **PROCESSING-INSTRUCTION-TARGET event (changed)**
 - **New parser:** If the processing instruction data is split into substrings, the PROCESSING-INSTRUCTION-TARGET event is repeated before each instance of the PROCESSING-INSTRUCTION-DATA event for a given processing instruction.
 - **Old parser:** The PROCESSING-INSTRUCTION-TARGET event occurs only once for a given processing instruction.
 - **To migrate to the new parser:** You might have to modify your code to accommodate multiple occurrences of the PROCESSING-INSTRUCTION-TARGET event while accumulating processing instruction data.

- **STANDALONE-DECLARATION event (no change)**
 - The old and new parsers have the same behavior for the STANDALONE-DECLARATION event.
 - **To migrate to the new parser:** No change required.
- **START-OF-CDATA-SECTION event (changed)**
 - **New parser:** All XML special registers except XML-EVENT, XML-CODE and XML-INFORMATION are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT always contains the string "`![CDATA]`".
 - **To migrate to the new parser:** If the string "`![CDATA]`" is acquired from the START-OF-CDATA-SECTION event, change your code to manually return it using a literal, or data item initialized with the value "`![CDATA]`".
- **START-OF-DOCUMENT event (changed)**
 - **New parser:** All XML special registers except XML-EVENT, XML-CODE and XML-INFORMATION are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entire document.
 - **To migrate to the new parser:** Change your code to not require the entire document for START-OF-DOCUMENT.
- **START-OF-ELEMENT event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the local part of the start element name or empty element name. If the element name is in a namespace, XML-NAMESPACE or XML-NNAMESPACE contains the namespace, otherwise these special registers are empty with length zero. If the element name is in a namespace and is prefixed (of the form "`prefix:local-part`"), XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix, otherwise these special registers are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the complete start element name, including any prefix. If the element name is not in a namespace, there is no difference between the old and new parsers for START-OF-ELEMENT.
 - **To migrate to the new parser:** If the element name is not in a namespace, then no change is required. If the element name is in a namespace, change your code to not use the complete element name, or reconstruct the complete element name from the separate parts in the XML text and namespace special registers.
- **UNKNOWN-REFERENCE-IN-ATTRIBUTE event (discontinued)**
 - **New parser:** Does not occur. The parser always signals an EXCEPTION event if, while processing an attribute value, it encounters a reference to an entity that has not been defined.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entity reference name, not including the "&" and ";" delimiters.
 - **To migrate to the new parser:** Ensure that your XML documents do not contain any undefined entity references in attribute values.
- **UNKNOWN-REFERENCE-IN-CONTENT event (discontinued)**
 - **New parser:** Does not occur. Instead, an UNRESOLVED-REFERENCE or EXCEPTION event occurs.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entity reference name, not including the "&" and ";" delimiters.

- **To migrate to the new parser:** Change your code that processes UNKNOWN-REFERENCE-IN-CONTENT to process UNRESOLVED-REFERENCE instead.

The UNRESOLVED-REFERENCE event is signaled only if all of the following conditions are true:

- The unresolved reference is within element content, not an attribute value.
- The XML document starts with an XML declaration that specifies standalone="no".
- The XML document contains a document type declaration, for example:
`<!DOCTYPE rootElementName>`
- If the VALIDATING phrase is specified on the XML PARSE statement, the document type declaration must also specify an external DTD subset, for example:
`<!DOCTYPE rootElementName SYSTEM "extSub.dtd">`

If these conditions are not met, the parser signals an EXCEPTION event instead of UNRESOLVED-REFERENCE.

- **UNRESOLVED-REFERENCE event (new)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the entity reference name, not including the "&" and ";" delimiters.
 - **Old parser:** The event does not occur. Instead an UNKNOWN-REFERENCE-IN-CONTENT event would occur.
 - **To migrate to the new parser:** See UNKNOWN-REFERENCE-IN-CONTENT.
- **VERSION-INFORMATION event (no change)**
 - both parsers have the same behavior for the VERSION-INFORMATION event.
 - **To migrate to the new parser:** No change required.

More differences between old and new parsers:

- XML-CODE
 - **New parser:** When XML-CODE is set by the parser for an EXCEPTION event, the first halfword is the return code and the last halfword is the reason code. Convert the value to hexadecimal. You can find common return code and reason code in the *z/OS XML System Services User's Guide and Reference*. You can also find COBOL specific return code and reason code in the *Enterprise COBOL Programming Guide*
 - **Old parser:** XML-CODE values are described in decimal in the *Enterprise COBOL Programming Guide, Version 4 Release 2*.
 - **To migrate to the new parser:** If your program tests for specific XML-CODE values for EXCEPTION events, you might have to change those values in your source program.
- Condition handling, RESUME, and XML PARSE statements
 - **New parser:** If a condition handling routine, registered by CEEHDLR or runtime option USERHDLR, gets control while executing a processing procedure due to an exception in the processing procedure and the resume cursor is moved by CEEMRCE to a point in the program before an XML PARSE statement, and RESUME is requested from the condition manager, the second XML PARSE would result in the following severity 3 runtime error message:

IGZ0228S There was an invalid attempt to start an XML PARSE statement.

- **Old parser:** If a condition handling routine (registered by CEEHDLR or runtime option USERHDLR) gets control while executing a processing procedure due to an exception in the processing procedure, and the resume cursor is moved by CEEMRCE to a point in the program before an XML PARSE statement, and RESUME is requested from the condition manage, the second XML PARSE would start successfully.
- **To migrate to the new parser:** Move the call to CEE3SRP to be within the processing procedure. Then at the resumption point, if the condition handling routine is unable to recover from the exception, terminate parsing by moving -1 to XML-CODE. If the condition handling routine is able to make an effective recovery, you might be able to continue parsing by leaving XML-CODE unchanged.

Alternatively, you can use CEEMRCR instead of CEEMRCE so that when execution is resumed, it is in the program that called the program that had the XML PARSE statement that got the exception in the processing procedure. Either of these methods properly addresses the exception.

The following table shows the predefined entity references.

Table 26. The predefined entity references

Predefined entity	Character
<	<
>	>
&	&
'	'
"	"

Upgrading Enterprise COBOL programs that have XML GENERATE statements

Enterprise COBOL introduced five new XML GENERATE exception codes after Enterprise COBOL Version 3.

Programs that use these exception codes might have to be changed to migrate to later versions of Enterprise COBOL.

The XML GENERATE exception codes that were added to Enterprise COBOL are:

- 415** The receiver was national, but the encoding specified for the document was not UTF-16.
- 416** The XML namespace identifier contained invalid XML characters.
- 417** Element character content or an attribute value contained characters that are illegal in XML content. XML generation has continued, with the element tag name or the attribute name prefixed with "hex." and the original data value represented in the document in hexadecimal.
- 418** Substitution characters were generated by encoding conversion.
- 419** The XML namespace prefix was invalid.

Converting programs that use new reserved words

Some reserved words have been added since Enterprise COBOL Version 3.

The new reserved words were:

- XML-INFORMATION
- XML-NAMESPACE
- XML-NAMESPACE-PREFIX
- XML-NNAMESPACE
- XML-NNAMESPACE-PREFIX
- XML-SCHEMA

The conversion tool CCCA automatically converts these reserved words for you if you have the PTF for APAR PM86253 installed. CCCA is included with the IBM Debug Tool product.

For a table comparing reserved words for all of the different COBOL compilers, see Table 40 on page 233.

Upgrading programs that use SIMVRD support

This section describes the actions to upgrade programs that use SIMVRD support. Support for *COBOL simulated variable-length relative-record data sets (RRDS)* is removed for programs compiled with Enterprise COBOL Version 4 or later. These files must be changed to VSAM RRDS files.

In COBOL compilers that supported the NOCMR2 compiler option before Enterprise COBOL Version 4, it was possible to use *COBOL simulated variable-length RRDS* using a VSAM KSDS when you used the SIMVRD runtime option support.

The coding that you use in a COBOL program to identify and describe VSAM variable-length RRDS and COBOL simulated variable-length RRDS is similar. With Enterprise COBOL Version 4 you must use VSAM variable-length RRDS support. In general, the only action to migrate from COBOL simulated variable-length RRDS to VSAM variable-length RRDS support is to change the IDCAMS definition of the file.

Table 27. Steps for using variable-length RRDS

Step	VSAM variable-length RRDS	COBOL simulated variable-length RRDS
1	Define the file with the ORGANIZATION IS RELATIVE clause.	Same
2	Use FD entries to describe the records with variable-length sizes.	Same, but you must also code RECORD IS VARYING in the FD entry of every COBOL program that accesses the data set.
3	Use the NOSIMVRD runtime option.	Use the SIMVRD runtime option.

Table 27. Steps for using variable-length RRDS (continued)

Step	VSAM variable-length RRDS	COBOL simulated variable-length RRDS
4	Define the VSAM file through access-method services as an RRDS.	Define the VSAM file through access-method services as follows: <pre> DEFINE CLUSTER INDEXED KEYS(4,0) RECORDSIZE(<i>avg</i>,<i>m</i>) </pre> <p><i>avg</i> Is the average size of the COBOL records; strictly less than <i>m</i>.</p> <p><i>m</i> Is greater than or equal to the maximum size COBOL record + 4.</p>

In step 2 for simulated variable-length RRDS, coding other language elements that implied a variable-length record format did not give you COBOL simulated variable-length RRDS. For example, the following elements alone did not cause the use of simulated variable-length RRDS access, and therefore did not require the SIMVRD runtime option:

- Multiple FD records of different lengths
- OCCURS . . . DEPENDING ON in the record definitions
- RECORD CONTAINS *integer-1* TO *integer-2* CHARACTERS

Use the REUSE IDCAMS parameter for files that contain records and that you will open for output.

- Define the file with the ORGANIZATION IS RELATIVE clause.
- Use FD entries to describe the records with variable-length sizes.
- Use the NOSIMVRD runtime option.
- Define the VSAM file through access-method services as an RRDS.

Errors: When you work with simulated variable-length relative data sets and true VSAM RRDS data sets, an OPEN file status 39 occurs if the COBOL file definition and the VSAM data-set attributes do not match.

For more reference information about the commands for using variable-length RRDS, see *z/OS DFSMS: Access Method Services for Catalogs*.

Chapter 12. Compiling Enterprise COBOL Version 3 programs

There have been a number of changes to compiler options and debug behavior for Enterprise COBOL Version 3 programs.

After reading these topics, see also Chapter 15, “Changes with IBM Enterprise COBOL for z/OS, Version 5.1,” on page 179.

Compiler option changes from IBM Enterprise COBOL for z/OS, Version 3

There have been a number of changes to compiler options.

The following options have been removed.

Table 28. Compiler options not available in Enterprise COBOL Version 5

Compiler option	Comments
DATEPROC	Support for Year 2000 extensions has been removed.
NOLIB	Compiler behaves as though LIB is always in effect.
YEARWINDOW	Support for Year 2000 extensions has been removed.
SIZE(MAX)	The SIZE option value is no longer an upper limit for the total storage used by a COBOL compilation. In addition, the SIZE suboption value MAX is no longer supported. The default value for the SIZE option is now SIZE(5000000).
NUMPROC(MIG)	NUMPROC(PFD) and NUMPROC(NOPFD) are still available. If NUMPROC(MIG) is specified, Enterprise COBOL, V5 issues a warning message and the compilation will get the default setting for NUMPROC. This is either the user-customized default or the IBM default, which is NUMPROC(NOPFD).

Also note, the SSRANGE compiled-in range checks cannot be disabled at run time using the runtime option CHECK(OFF) or NOSSRANGE

For descriptions of new and modified options for Enterprise COBOL V5.1, see “Compiler option changes in IBM Enterprise COBOL for z/OS, Version 5” on page 181.

For a detailed list of options supported for the various compiler versions, see Appendix E, “Option comparison,” on page 263.

Differences in the TEST compiler option

This section provides information about a simplified TEST compiler option you need to know when you upgrade programs compiled with the TEST compiler option. Enterprise COBOL Version 5 has a simplified TEST compiler option compared to earlier compilers. If the TEST option is specified in JCL or CBL/PROCESS statements in COBOL source, you may want to change them. The following TEST suboptions are deprecated, but some continue to be tolerated to

ease migration. Compiler diagnostics messages are issued if they are used. The deprecated suboptions may not be specified together with new suboptions in the same TEST option specification.

Table 29. The deprecated TEST suboptions

Deprecated suboption	Behavior if specified with compiler	Diagnostic message level or category
ALL	Debugging information generated	Error (Invalid option diagnostic, option discarded)
BLOCK	Debugging information generated	Error (Invalid option diagnostic, option discarded)
PATH	Debugging information generated	Error (Invalid option diagnostic, option discarded)
STMT	Debugging information generated	Error (Invalid option diagnostic, option discarded)
NONE		Error (Invalid option diagnostic, option discarded)
SYM	Ignored, symbolic debugging information always generated	Error (Invalid option diagnostic, option discarded)
NOSYM	Ignored, symbolic debugging information always generated	Error (Invalid option diagnostic, option discarded)
HOOK	Ignored, symbolic debugging information always generated	Informational message about NOHOOK behavior always in effect (Suboption tolerated, TEST in effect)
NOHOOK	Ignored, symbolic debugging information always generated	Informational message about NOHOOK behavior always in effect (Suboption tolerated, TEST in effect)
SEPARATE	Ignored, symbolic debugging information always generated	Informational message about NOSEPARATE behavior always in effect (Suboption tolerated, TEST in effect)
NOSEPARATE	Ignored, symbolic debugging information always generated	Informational message about NOSEPARATE behavior always in effect (Suboption tolerated, TEST in effect)

Note: The HOOK and SEPARATE suboptions are only deprecated as invocation options or on CBL/PROCESS statements. They are not recognized at all in IGYCDOPT for setting installation default options.

Debug information changes with IBM Enterprise COBOL Version 5

Programs compiled with IBM Enterprise COBOL Version 5 will have different debug information than that of programs compiled with previous versions of the compiler.

IBM Enterprise COBOL Version 5 solves the dilemma of debugging information. In the past you had 2 choices:

- Have the debug data always with the executable at a cost of a large load footprint, or
- Have separate debug data but also have the challenge of keeping it synchronized with the application and finding it when needed.

Now you have the best of both worlds. With NOLOAD debug segments in the program object, the debug data does not increase the size of the loaded program, it always matches the executable and is always available so there is no need to search lists of data sets.

There have been changes to the TEST compiler option used to generate debuggable versions of your application and to the NOTEST option.

- When the TEST option is specified, DWARF debug information is included in the application module.
- If the SOURCE suboption is specified, the DWARF debug information includes the expanded source code, and the compiler listing is not needed by IBM Debug Tool. When the TEST(NOSOURCE) compiler option is specified, the generated DWARF debugging information does not include the expanded source code.
- You can use the NOTEST(DWARF) compiler option to include basic DWARF debugging information in the program object. You cannot debug such programs with Debug Tool, but you can get NOTEST optimization and still enable application failure analysis tools, such as CEEDUMP output and IBM Fault Analyzer.
- To have no debugging information in the program object, use the NOTEST(NODWARF) option.

When debugging your COBOL programs, you will find that there have been a large number of improvements and behavior changes introduced with Enterprise COBOL V5.1. For details about changes in debugging with IBM Debug Tool, see “Debug Tool changes with IBM Enterprise COBOL Version 5” on page 204.

Chapter 13. Upgrading from Enterprise COBOL Version 4

To compile with Enterprise COBOL V5, programs that use any of several features must be upgraded.

Programs that contain any of the following language features need to be modified:

1. Programs that use XML PARSE with XMLPARSE(COMPAT)
2. V4R1 programs that use XML PARSE with XMLPARSE(XMLSS)
3. Programs using DATE FORMAT and windowed date functions. For details, see “Changes in millenium language extensions in IBM Enterprise COBOL for z/OS, Version 5” on page 174.
4. Label declaratives. To compile programs with Enterprise COBOL V5, you must remove any format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE..., and the syntax: GO TO MORE-LABELS. The support for these were removed in Enterprise COBOL Version 5
5. Programs that use new reserved words as user words. For details, see “New reserved words” on page 96.

There is a new compiler option, FLAGMIG4, available with APAR PM93450 for Enterprise COBOL V4.2 to help you migrate to Enterprise COBOL V5. The FLAGMIG4 option identifies language elements in Enterprise COBOL V4 programs that are not supported, or that are supported differently in Enterprise COBOL V5. The compiler will generate a warning diagnostic message for all such language elements.

Tip: It is recommended that you review and apply the Enterprise COBOL V4 PTFs to support the migration to Enterprise COBOL V5 or V6. For details, see <http://www.ibm.com/support/docview.wss?uid=swg21982146>.

Upgrading Enterprise COBOL Version 4 programs that have XML PARSE statements

If you were using the z/OS System Services XML parser with your COBOL Version 4.2 compiler, you do not need to make any code changes for Enterprise COBOL Version 5.

Enterprise COBOL V5 no longer requires the XMLPARSE option since it only supports the z/OS System Services XML parser. You can no longer use the XMLPARSE(COMPAT) option to use the old XML parser that was part of the COBOL runtime library.

If you were using the old parser and the XMLPARSE(COMPAT) option, you will have to migrate to the new parser. For information on the new, changed, and discontinued XML parsing events, see “Migrating from the old XML parser to the new XML parser” on page 150.

If you were using the z/OS System Services XML parser with COBOL Version 4.1, you should consider information in “Upgrading Enterprise COBOL Version 4 Release 1 programs that have XML PARSE statements and that use the XMLPARSE(XMLSS) compiler option” on page 173.

Migrating from the old XML parser to the new XML parser

New, changed, unchanged, and discontinued events with XML PARSE in Enterprise COBOL V5:

- You can migrate your programs to use the z/OS System Services XML parser (referred to as the *new parser*) from the XML parser that was part of the COBOL runtime environment (referred to as the *old parser*) after you understand the differences between the new and the old parsers. Some of these differences are described in terms of new, changed, unchanged, and discontinued events when the new parser is in effect:
 - **ATTRIBUTE-CHARACTER event (discontinued)**
 - **New parser:** The ATTRIBUTE-CHARACTER event no longer occurs. All entity references, including predefined ones, are now included in the ATTRIBUTE-CHARACTERS event, unless there is an unresolved entity reference, in which case an EXCEPTION event is signaled.
 - **Old parser:** The ATTRIBUTE-CHARACTER event occurs for predefined entity references only. The five predefined entity references are shown in Table 26 on page 157. XML-TEXT or XML-NTEXT contains the single character that corresponds with the predefined entity reference in the attribute value. Character references are signaled as ATTRIBUTE-NATIONAL-CHARACTER events.
 - **To migrate to the new parser:** Remove references to the ATTRIBUTE-CHARACTER event and integrate any actions for this event into your ATTRIBUTE-CHARACTERS event handling.
 - **ATTRIBUTE-CHARACTERS event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT could have a substring of the value for the ATTRIBUTE-CHARACTERS event. XML-TEXT or XML-NTEXT could also contain a complete string of the value even if the value contains a character reference or an entity reference.
 - **Old parser:** XML-TEXT or XML-NTEXT has only a substring of the value for the ATTRIBUTE-CHARACTERS event when the value contains a character reference or an entity reference.
 - **To migrate to the new parser:** You might have to modify your code that handles the ATTRIBUTE-CHARACTERS event to handle more than one event even if your attribute values do not contain character or entity references. You might also have to change your code to process ATTRIBUTE-CHARACTERS as a single event where your code was handling ATTRIBUTE-CHARACTERS as multiple events.
 - **ATTRIBUTE-NAME event (changed)**
 - **New parser:** For attribute names that are not in a namespace, XML-TEXT or XML-NTEXT contains the attribute name, and the namespace special registers are all empty and have length zero. Attributes with names in a namespace are always prefixed and have the form:
prefix:local-part = AttValue

XML-TEXT or XML-NTEXT contains the local-part, XML-NAMESPACE or XML-NNAMESPACE contains the namespace and XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix.
 - **Old parser:** For all attribute names, XML-TEXT or XML-NTEXT contains the complete attribute name, even if the name is prefixed (indicating that the name belongs to a namespace).

- **To migrate to the new parser:** Either change your code to process the separate parts of the namespace, or change your code to reconstruct the complete attribute name from the separate parts in XML-TEXT, XML-NAMESPACE-PREFIX, and XML-NAMESPACE, or XML-NTEXT, XML-NNAMESPACE-PREFIX, and XML-NNAMESPACE.
- **ATTRIBUTE-NATIONAL-CHARACTER event (changed)**
 - **New parser:** Character references that can be represented in the EBCDIC encoding of the XML document are resolved and included in the ATTRIBUTE-CHARACTERS event.
Unrepresentable character references are expressed as ATTRIBUTE-NATIONAL-CHARACTER events, as for COMPAT.
 - **Old parser:** Regardless of the type of the XML document specified by identifier-1 in the XML PARSE statement, XML-TEXT is empty and XML-NTEXT contains the single national character corresponding with the (numeric) character reference.
 - **To migrate to the new parser:** Possibly no change will be required, but be aware that with the old parser, the national character might have an EBCDIC equivalent, whereas with the new parser, the national character is known to have no representation in the EBCDIC encoding of the document.
- **COMMENT event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT could have a substring of the value for the COMMENT event.
 - **Old parser:** XML-TEXT or XML-NTEXT always has the complete string of the value for the COMMENT event.
 - **To migrate to the new parser:** You might have to modify your code that handles the COMMENT event to handle more than one event if you get a substring of the COMMENT value in XML-TEXT or XML-NTEXT. If that is the case, you get two or more COMMENT events in succession and you would concatenate strings together to re-create the complete string of the value. You cannot distinguish a comment that is split in this way from a sequence of distinct comments.
- **CONTENT-CHARACTER event (discontinued)**
 - **New parser:** The CONTENT-CHARACTER event no longer occurs. All entity references, including predefined ones, are now included in the CONTENT-CHARACTERS event unless there is an unresolved entity reference, in which case an UNRESOLVED-REFERENCE event or an EXCEPTION event is signaled.
 - **Old parser:** The CONTENT-CHARACTER event occurs for predefined entity references only. The five predefined entity references are shown in Table 26 on page 157. XML-TEXT or XML-NTEXT contains the single character that corresponds with the predefined entity reference in the attribute value. Character references are signaled as CONTENT-NATIONAL-CHARACTER events.
 - **To migrate to the new parser:** Remove references to the CONTENT-CHARACTER event and integrate any actions for this event into your CONTENT-CHARACTERS event handling.
- **CONTENT-CHARACTERS event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT could have a substring of the content for the CONTENT-CHARACTERS event. XML-TEXT or XML-NTEXT could also contain a complete string of the content even if the content contains a character reference or an entity reference.

- **Old parser:** XML-TEXT or XML-NTEXT has only a substring of the content for the CONTENT-CHARACTERS event when the content contains a character reference or an entity reference.
- **To migrate to the new parser:** You might have to modify your code that handles the CONTENT-CHARACTERS event to handle more than one event even if your attribute values do not contain character or entity references. You might also have to change your code to process CONTENT-CHARACTERS as a single event where your code was handling CONTENT-CHARACTERS as multiple events.
- **CONTENT-NATIONAL-CHARACTER event (changed)**
 - **New parser:** Character references that can be represented in the EBCDIC encoding of the XML document are resolved and included in the CONTENT-CHARACTERS event.
Unrepresentable character references are expressed as CONTENT-NATIONAL-CHARACTER events, as for COMPAT.
 - **Old parser:** Regardless of the type of the XML document specified by identifier-1 in the XML PARSE statement, XML-TEXT is empty, and XML-NTEXT contains the single national character corresponding with the (numeric) character reference.
 - **To migrate to the new parser:** Possibly no change will be required, but be aware that with the old parser, the national character might have an EBCDIC equivalent, whereas with the new parser, the national character is known to have no representation in the EBCDIC encoding of the document.
- **DOCUMENT-TYPE-DECLARATION event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the name of the root element, as specified in the document type declaration. The parser processes entity declarations and default attribute values in the internal DTD subset, and ignores the rest of the text in the document type declaration.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entire document type declaration.
 - **To migrate to the new parser:** If having the whole document type declaration is important, you might have to modify your code that handles the DOCUMENT-TYPE-DECLARATION event to acquire the information directly from your XML document.
- **ENCODING-DECLARATION event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the encoding name. The encoding declaration is not used by the parser, so you might get incorrect characters passed through that would cause the parser to signal an EXCEPTION event from which you can't recover.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the encoding name. If there are errors in the encoding of the document, you would get an EXCEPTION event from which you might be able to recover and continue.
 - **To migrate to the new parser:** Check your document before parsing or specify your encoding using the CODEPAGE compiler option or by using the WITH ENCODING phrase on the XML PARSE statement.
- **END-OF-CDATA-SECTION event (changed)**
 - **New parser:** All XML special registers except XML-EVENT, XML-CODE and XML-INFORMATION are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT always contains the string "]]>".

- **To migrate to the new parser:** If the string "]]>" is acquired from the END-OF-CDATA-SECTION event, change your code to manually return it using a literal, or data item initialized with the value "]]>".
- **END-OF-DOCUMENT event (no change)**
 - The 2 parsers have the same behavior for the END-OF-DOCUMENT event.
 - **To migrate to the new parser:** No change required.
- **END-OF-ELEMENT event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the local part of the end element tag or empty element tag name. If the element name is in a namespace, XML-NAMESPACE or XML-NNAMESPACE contains the namespace, otherwise these special registers are empty with length zero. If the element name is in a namespace and is prefixed (of the form "prefix:local-part"), XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix, otherwise these special registers are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the complete element tag name, including any prefix. If the element name is not in a namespace, there is no difference between the old and new parsers for END-OF-ELEMENT.
 - **To migrate to the new parser:** If the element name is not in a namespace, then no change is required. If the element name is in a namespace, change your code to not use the complete element name, or reconstruct the complete element name from the separate parts in the XML text and namespace special registers.
- **END-OF-INPUT event (new)**
 - **New parser:** The END-OF-INPUT event indicates the end of a segment of an XML document.
 - **Old parser:** The END-OF-INPUT event does not occur.
 - **To migrate to the new parser:** With the old parser, your document is in one segment, so no change is required to change to the new parser.
- **EXCEPTION event (changed)**
 - **New parser:** XML-CODE contains the unique return code and reason code identifying the exception. See the following section "Other differences" for a description of XML-CODE differences. XML-TEXT or XML-NTEXT contains the document fragment up to the point of the error or anomaly that caused the EXCEPTION event. All other XML special registers except XML-EVENT and XML-INFORMATION are empty with length zero. It is not possible to continue from any EXCEPTION event.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entire document that has been parsed up to the point of the EXCEPTION event. It is possible to continue from some EXCEPTION events.
 - **To migrate to the new parser:** You might have to change your code or documents if they depend on being able to recover from EXCEPTION events.
- **NAMESPACE-DECLARATION event (new)**
 - **New parser:** XML-TEXT and XML-NTEXT are both empty with length zero. XML-NAMESPACE or XML-NNAMESPACE contains the declared namespace. If the namespace is "undeclared" by specifying the empty string, XML-NAMESPACE and XML-NNAMESPACE are empty with length zero. XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix if the attribute name for the namespace declaration is of the form "xmlns:prefix", otherwise, if the declaration is for the default

namespace and the attribute name is "xmlns", XML-NAMESPACE-PREFIX and XML-NNAMESPACE-PREFIX are both empty with length zero.

- **Old parser:** The NAMESPACE-DECLARATION event does not occur.
- **To migrate to the new parser:** If you get the NAMESPACE-DECLARATION event after migrating to the new parser, see the descriptions in this table of ATTRIBUTE-NAME, END-OF-ELEMENT and START-OF-ELEMENT event changes.
- **PROCESSING-INSTRUCTION-DATA event (changed)**
 - **New parser:** XML-TEXT or XML-NTEXT could have a substring of the value for the PROCESSING-INSTRUCTION-DATA event.
 - **Old parser:** XML-TEXT or XML-NTEXT always has the complete string of the value for the PROCESSING-INSTRUCTION-DATA event.
 - **To migrate to the new parser:** You might have to modify your code that handles the PROCESSING-INSTRUCTION-DATA event to handle more than one event if you get a substring of the PROCESSING-INSTRUCTION-DATA value in XML-TEXT or XML-NTEXT. If that is the case, you get two or more PROCESSING-INSTRUCTION-DATA events, each one preceded by its matching PROCESSING-INSTRUCTION-TARGET event. You would then concatenate the PROCESSING-INSTRUCTION-DATA substrings together to reconstitute the complete data string.
- **PROCESSING-INSTRUCTION-TARGET event (changed)**
 - **New parser:** If the processing instruction data is split into substrings, the PROCESSING-INSTRUCTION-TARGET event is repeated before each instance of the PROCESSING-INSTRUCTION-DATA event for a given processing instruction.
 - **Old parser:** The PROCESSING-INSTRUCTION-TARGET event occurs only once for a given processing instruction.
 - **To migrate to the new parser:** You might have to modify your code to accommodate multiple occurrences of the PROCESSING-INSTRUCTION-TARGET event while accumulating processing instruction data.
- **STANDALONE-DECLARATION event (no change)**
 - The old and new parsers have the same behavior for the STANDALONE-DECLARATION event.
 - **To migrate to the new parser:** No change required.
- **START-OF-CDATA-SECTION event (changed)**
 - **New parser:** All XML special registers except XML-EVENT, XML-CODE and XML-INFORMATION are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT always contains the string "![CDATA[".
 - **To migrate to the new parser:** If the string "![CDATA[" is acquired from the START-OF-CDATA-SECTION event, change your code to manually return it using a literal, or data item initialized with the value "![CDATA[".
- **START-OF-DOCUMENT event (changed)**
 - **New parser:** All XML special registers except XML-EVENT, XML-CODE and XML-INFORMATION are empty with length zero.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entire document.
 - **To migrate to the new parser:** Change your code to not require the entire document for START-OF-DOCUMENT.
- **START-OF-ELEMENT event (changed)**

- **New parser:** XML-TEXT or XML-NTEXT contains the local part of the start element name or empty element name. If the element name is in a namespace, XML-NAMESPACE or XML-NNAMESPACE contains the namespace, otherwise these special registers are empty with length zero. If the element name is in a namespace and is prefixed (of the form "prefix:local-part"), XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX contains the prefix, otherwise these special registers are empty with length zero.
- **Old parser:** XML-TEXT or XML-NTEXT contains the complete start element name, including any prefix. If the element name is not in a namespace, there is no difference between the old and new parsers for START-OF-ELEMENT.
- **To migrate to the new parser:** If the element name is not in a namespace, then no change is required. If the element name is in a namespace, change your code to not use the complete element name, or reconstruct the complete element name from the separate parts in the XML text and namespace special registers.
- **UNKNOWN-REFERENCE-IN-ATTRIBUTE event (discontinued)**
 - **New parser:** Does not occur. The parser always signals an EXCEPTION event if, while processing an attribute value, it encounters a reference to an entity that has not been defined.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entity reference name, not including the "&" and ";" delimiters.
 - **To migrate to the new parser:** Ensure that your XML documents do not contain any undefined entity references in attribute values.
- **UNKNOWN-REFERENCE-IN-CONTENT event (discontinued)**
 - **New parser:** Does not occur. Instead, an UNRESOLVED-REFERENCE or EXCEPTION event occurs.
 - **Old parser:** XML-TEXT or XML-NTEXT contains the entity reference name, not including the "&" and ";" delimiters.
 - **To migrate to the new parser:** Change your code that processes UNKNOWN-REFERENCE-IN-CONTENT to process UNRESOLVED-REFERENCE instead.

The UNRESOLVED-REFERENCE event is signaled only if all of the following conditions are true:

 - The unresolved reference is within element content, not an attribute value.
 - The XML document starts with an XML declaration that specifies standalone="no".
 - The XML document contains a document type declaration, for example:

```
<!DOCTYPE rootElementName>
```
 - If the VALIDATING phrase is specified on the XML PARSE statement, the document type declaration must also specify an external DTD subset, for example:

```
<!DOCTYPE rootElementName SYSTEM "extSub.dtd">
```

If these conditions are not met, the parser signals an EXCEPTION event instead of UNRESOLVED-REFERENCE.
- **UNRESOLVED-REFERENCE event (new)**
 - **New parser:** XML-TEXT or XML-NTEXT contains the entity reference name, not including the "&" and ";" delimiters.

- **Old parser:** The event does not occur. Instead an UNKNOWN-REFERENCE-IN-CONTENT event would occur.
- **To migrate to the new parser:** See UNKNOWN-REFERENCE-IN-CONTENT.
- **VERSION-INFORMATION event (no change)**
 - both parsers have the same behavior for the VERSION-INFORMATION event.
 - **To migrate to the new parser:** No change required.

More differences between old and new parsers:

- XML-CODE
 - **New parser:** When XML-CODE is set by the parser for an EXCEPTION event, the first halfword is the return code and the last halfword is the reason code. Convert the value to hexadecimal. You can find common return code and reason code in the *z/OS XML System Services User's Guide and Reference*. You can also find COBOL specific return code and reason code in the *Enterprise COBOL Programming Guide*
 - **Old parser:** XML-CODE values are described in decimal in the *Enterprise COBOL Programming Guide, Version 4 Release 2*.
 - **To migrate to the new parser:** If your program tests for specific XML-CODE values for EXCEPTION events, you might have to change those values in your source program.
- Condition handling, RESUME, and XML PARSE statements
 - **New parser:** If a condition handling routine, registered by CEEHDLR or runtime option USERHDLR, gets control while executing a processing procedure due to an exception in the processing procedure and the resume cursor is moved by CEEMRCE to a point in the program before an XML PARSE statement, and RESUME is requested from the condition manager, the second XML PARSE would result in the following severity 3 runtime error message:
 IGZ0228S There was an invalid attempt to start an XML PARSE statement.
 - **Old parser:** If a condition handling routine (registered by CEEHDLR or runtime option USERHDLR) gets control while executing a processing procedure due to an exception in the processing procedure, and the resume cursor is moved by CEEMRCE to a point in the program before an XML PARSE statement, and RESUME is requested from the condition manager, the second XML PARSE would start successfully.
 - **To migrate to the new parser:** Move the call to CEE3SRP to be within the processing procedure. Then at the resumption point, if the condition handling routine is unable to recover from the exception, terminate parsing by moving -1 to XML-CODE. If the condition handling routine is able to make an effective recovery, you might be able to continue parsing by leaving XML-CODE unchanged.

 Alternatively, you can use CEEMRCR instead of CEEMRCE so that when execution is resumed, it is in the program that called the program that had the XML PARSE statement that got the exception in the processing procedure. Either of these methods properly addresses the exception.

The following table shows the predefined entity references.

Table 30. The predefined entity references

Predefined entity	Character
<	<
>	>
&	&
'	'
"	"

Upgrading Enterprise COBOL Version 4 Release 1 programs that have XML PARSE statements and that use the XMLPARSE(XMLSS) compiler option

There are differences in XML PARSE behavior with the XMLPARSE(XMLSS) compiler option in effect between Enterprise COBOL Version 4 Release 1 and Enterprise COBOL Version 4 Release 2 or later. In Enterprise COBOL Version 4 Release 1 when you parsed an XML document using the XMLPARSE(XMLSS) compiler option and it contained character references that could not be expressed in the encoding of the document, the result was a single ATTRIBUTE-CHARACTERS or CONTENT-CHARACTERS XML event in which every unrepresentable character reference was replaced by a hyphen-minus. No indication was given to the program that the substitution occurred.

For example, parsing the content of the following XML element:

```
<elem>abc&#x1234;xyz</elem>
```

under Enterprise COBOL Version 4 Release 1 with encoding CCSID 1140 and with the XMLPARSE(XMLSS) compiler option in effect, resulted in a single CONTENT-CHARACTERS XML event with special register XML-TEXT containing the (EBCDIC) string:

```
abc-xyz
```

and with special register XML-CODE containing zero.

In Enterprise COBOL Version 4 Release 2 and later, when you parse an XML document using the XMLPARSE(XMLSS) compiler option, instead of a single ATTRIBUTE-CHARACTERS or CONTENT-CHARACTERS event, multiple XML events occur. Each unrepresentable character reference previously replaced by a hyphen-minus is instead expressed as an ATTRIBUTE-NATIONAL-CHARACTER or CONTENT-NATIONAL-CHARACTER XML event, depending on the context in which it occurred. These are new XML events for the XMLPARSE(XMLSS) compiler option.

Parsing the content of the XML element from before:

```
<elem>abc&#x1234;xyz</elem>
```

under Enterprise COBOL Version 4 Release 2 results in the following sequence of XML events:

- CONTENT-CHARACTERS with XML-TEXT containing abc
- CONTENT-NATIONAL-CHARACTER with XML-NTEXT containing NX'1234'
- CONTENT-CHARACTERS with XML-TEXT containing xyz

Changes in millenium language extensions in IBM Enterprise COBOL for z/OS, Version 5

The Millennium Language Extensions are no longer supported.

The elements that have been removed are:

- DATE FORMAT clause
- DATEVAL intrinsic function
- UNDATE intrinsic function
- YEARWINDOW intrinsic function
- DATEPROC compiler option
- YEARWINDOW compiler option

These language elements must be removed in order to compile with Enterprise COBOL V5.1

Chapter 14. Compiling Enterprise COBOL V4 programs

There have been a number of changes to compiler options and debug behavior for Enterprise COBOL Version 4 programs.

After reading these topics, see also Chapter 15, “Changes with IBM Enterprise COBOL for z/OS, Version 5.1,” on page 179.

Compiler option changes from IBM Enterprise COBOL for z/OS, Version 4

There have been a number of changes to compiler options.

The following options have been deprecated.

Table 31. Compiler options not available in Enterprise COBOL Version 5

Compiler option	Comments
DATEPROC	Support for Year 2000 extensions has been removed.
NOLIB	Compiler behaves as though LIB is always in effect.
YEARWINDOW	Support for Year 2000 extensions has been removed.
SIZE(MAX)	The SIZE option value is no longer an upper limit for the total storage used by a COBOL compilation. In addition, the SIZE suboption value MAX is no longer supported. The default value for the SIZE option is now SIZE(5000000).
XMLPARSE	The compiler behaves as though the XMLPARSE(XMLSS) compiler option is always in affect
NUMPROC(MIG)	NUMPROC(PFD) and NUMPROC(NOPFD) are still available. If NUMPROC(MIG) is specified, Enterprise COBOL, V5 issues a warning message and the compilation will get the default setting for NUMPROC. This is either the user-customized default or the IBM default, which is NUMPROC(NOPFD).

Also note, the compiled-in range checks cannot be disabled at run time using the runtime option CHECK(OFF) or NOSSRANGE

For descriptions of new and modified options for Enterprise COBOL Version 5, see “Compiler option changes in IBM Enterprise COBOL for z/OS, Version 5” on page 181.

For a detailed list of options supported for the various compiler versions, see Appendix E, “Option comparison,” on page 263.

For detailed descriptions of all options, see the *Enterprise COBOL Programming Guide*.

Debug information changes with IBM Enterprise COBOL Version 5

Programs compiled with IBM Enterprise COBOL Version 5 will have different debug information than that of programs compiled with previous versions of the compiler.

IBM Enterprise COBOL Version 5 solves the dilemma of debugging information. In the past you had 2 choices:

- Have the debug data always with the executable at a cost of a large load footprint, or
- Have separate debug data but also have the challenge of keeping it synchronized with the application and finding it when needed.

Now you have the best of both worlds. With NOLOAD debug segments in the program object, the debug data does not increase the size of the loaded program, it always matches the executable and is always available so there is no need to search lists of data sets.

There have been changes to the TEST compiler option used to generate debuggable versions of your application and to the NOTEST option.

- When the TEST option is specified, DWARF debug information is included in the application module.
- If the SOURCE suboption is specified, the DWARF debug information includes the expanded source code, and the compiler listing is not needed by IBM Debug Tool. When the TEST(NOSOURCE) compiler option is specified, the generated DWARF debugging information does not include the expanded source code.
- You can use the NOTEST(DWARF) compiler option to include basic DWARF debugging information in the program object. You cannot debug such programs with Debug Tool, but you can get NOTEST optimization and still enable application failure analysis tools, such as CEEDUMP output and IBM Fault Analyzer.
- To have no debugging information in the program object, use the NOTEST(NODWARF) option.

When debugging your COBOL programs, you will find that there have been a large number of improvements and behavior changes introduced with Enterprise COBOL V5.1. For details about changes in debugging with IBM Debug Tool, see “Debug Tool changes with IBM Enterprise COBOL Version 5” on page 204.

Part 4. What is new and different with Enterprise COBOL V5.1?

There are a few differences from all previous compilers to consider when using Enterprise COBOL V5.1. After reading the section about migrating a program or application from the compiler you are currently using, read this section.

Chapter 15. Changes with IBM Enterprise COBOL for z/OS, Version 5.1

There are many changes to the compiler and runtime library with Enterprise COBOL V5.1. There are changes in compiling, link editing/binding, execution, and even changed Debug Tool behavior.

These changes fall into the following categories:

1. Prerequisite software and service
2. Source code differences
3. Compiler option differences
4. Compiling behavior differences
5. Program management binder (Link edit)
6. Differences at run time
7. Debug information changes

Prerequisite software and service for Enterprise COBOL V5

Updates are required for other products to compile programs with Enterprise COBOL V5 and also to bind, run and debug those programs. Now, with Enterprise COBOL V5, you can use FIXCAT to find required service.

Prerequisite levels of related software products

To use these products with Enterprise COBOL V5, they must be at the following levels:

- z/OS V1R13 or later
- CICS Transaction Server for z/OS, V3 or later
- IBM DB2 V9 or later
- IBM IMS V11 or later
- PD Tools V12 or later (Debug Tool, Fault Analyzer, Application Performance Analyzer)
- Rational Developer for System z V9 (RDz) or later

Determining service required

You no longer need to find lists of APARs and PTFs in PSP buckets. As of Enterprise COBOL for z/OS, V5.1, you must use SMP/E FIXCATs to identify the required PTFs on other products to work with Enterprise COBOL for z/OS, V5.1. The required service PTFs for COBOL for z/OS V5.1 are not documented in this Migration Guide, are not included in PSP buckets, and are not included in any handouts for conferences.

SMP/E FIXCATs allow you to have the most up to date and correct information about Enterprise COBOL for z/OS, V5.1 required service. It is the easiest way to quickly determine if you have all the necessary required service PTFs installed. For Enterprise COBOL for z/OS, V5.1, you should use SMP/E V3R5 or later support for FIXCAT HOLDDATA to do programmatic target system PTF verification. These

PTFs are identified with a FIXCAT called IBM.TargetSystem-RequiredService.Enterprise-COBOL.V5R1 in Enhanced HOLDDATA.

A HOLDDATA type FIXCAT (fix category) is used to associate an APAR to a particular category of fix for necessary target system PTFs. To help identify PTFs required but not yet installed for your upgrade to Enterprise COBOL for z/OS, V5R1 on your current system, use the SMP/E **REPORT MISSINGFIX** command. Here is a sample command used to run against your z/OS CSI:

```
SET BDY(GLOBAL).  
REPORT MISSINGFIX ZONES(ZOS13T)  
FIXCAT(IBM.TargetSystem-RequiredService.Enterprise-COBOL.V5R1)
```

For complete information about the **REPORT MISSINGFIX** command, see *SMP/E Commands*.

Enterprise COBOL V4.2 aids for migration to Enterprise COBOL V5

Fixes for previous versions of Enterprise COBOL are not handled by FIXCAT. The following APAR fixes contain aids for helping you migrate from Enterprise COBOL V4.2 to Enterprise COBOL V5.

- PM93450 - FLAGMIG4. This one helps you identify if you have COBOL statements that are unsupported in V5.
- PM85035 - new function to support XML-INFORMATION special register. This one helps you migrate to XMLPARSE(XMLSS) and therefore to V5
- Language Environment, V1.13 PM87347 for XML-INFORMATION support at run time if you have installed the related Enterprise COBOL V4 APAR, PM85035.

COBOL source code differences in Enterprise COBOL V5.1

Several language elements have been removed or modified in IBM Enterprise COBOL for z/OS, V5.1 that may require updates to your source programs.

The Millennium Language Extensions are no longer supported. If your programs have any of these language elements, they must be removed before you can compile and run these programs with Enterprise COBOL V5 :

- DATE FORMAT clause
- DATEVAL intrinsic function
- UNDATE intrinsic function
- YEARWINDOW intrinsic function

There have been changes to LABEL declarative support. If your programs have any of these language elements, they must be removed before you can compile and run these programs with Enterprise COBOL V5 :

- Format 2 declarative syntax: USE...AFTER...LABEL PROCEDURE... is no longer supported
- The syntax: GO TO MORE-LABELS is no longer supported.

Compiler option changes in IBM Enterprise COBOL for z/OS, Version 5

There have been a number of changes to compiler options.

The following options have been deprecated.

Table 32. Compiler options not available in Enterprise COBOL Version 5

Compiler option	Comments
DATEPROC	Support for Year 2000 extensions has been removed.
NOLIB	Compiler behaves as though LIB is always in effect.
YEARWINDOW	Support for Year 2000 extensions has been removed.
SIZE(MAX)	The MAX suboption of the SIZE option is no longer supported.
XMLPARSE	The compiler behaves as though the XMLPARSE(XMLSS) compiler option is always enabled.
NUMPROC(MIG)	NUMPROC(PFD) and NUMPROC(NOPFD) are still available. If NUMPROC(MIG) is specified, Enterprise COBOL, V5 issues a warning message and the compilation will get the default setting for NUMPROC. This is either the user-customized default or the IBM default, which is NUMPROC(NOPFD).

The following options have been added or modified.

Table 33. Compiler options new and changed with Enterprise COBOL Version 5

Compiler option	Comments
AFP	New option. It controls the compiler usage of the Additional Floating Point (AFP) registers that are provided by z/Architecture processors. AFP(VOLATILE) is the default.
ARCH	New option. It specifies the machine architecture for which the executable program instructions are to be generated. ARCH(6) is the default.
DISPSIGN	New option. It controls output formatting for DISPLAY of signed numeric items. DISPSIGN(COMPAT) is the default.
HGPR	New option. It controls the compiler usage of the 64-bit registers provided by z/Architecture processors. HGPR(PRESERVE) is the default.
MAXPCF	New option. It instructs the compiler not to optimize code if the program contains a complexity factor greater than <i>n</i> .
SIZE	The SIZE option value is no longer an upper-limit for the total storage used by a COBOL compilation. In addition, the SIZE suboption value MAX is no longer supported. The default value for the SIZE option is now SIZE(5000000). For more information about compiler memory requirements, see “Changes in compiling with Enterprise COBOL Version 5.1” on page 182
STGOPT	New option. It controls storage optimization. NOSTGOPT is the default.
EXIT	The EXIT compiler option is no longer mutually exclusive with the DUMP compiler option, and the compiler exits rules are updated.
MDECK	The MDECK option no longer has a dependency on the LIB option, as the compiler behaves as though the LIB option is always enabled.
NORENT	NORENT can no longer be used with RMODE(ANY). Execution of NORENT programs above the 16 MB line is not supported.

Table 33. Compiler options new and changed with Enterprise COBOL Version 5 (continued)

Compiler option	Comments
RMODE(ANY)	RMODE(ANY) can no longer be used with NORENT
SSRANGE	The compiled-in range checks cannot be disabled at run time using the runtime options CHECK(OFF) or NOSSRANGE. Note: the compiler option NOSSRANGE is still supported.
TEST	<p>The HOOK NOHOOK and SEPARATE NOSEPARATE suboptions of the TEST compiler option are deprecated. If specified,</p> <ul style="list-style-type: none"> • HOOK - compiled in hooks are not available. • NOHOOK - NOHOOK behavior is always in effect • SEPARATE - Compiler always places debugging info in object • NOSEPARATE - NOSEPARATE behavior is always in effect <p>New suboptions SOURCE and NOSOURCE are added to the TEST compiler option.</p> <p>Note: EJPD and NOEJPD suboptions are still supported. With Debug Tool V12 with APAR PM75819 or Debug Tool V13 or later, you can do JUMPTO or GOTO even if you compile with the TEST(NOJPD) option and a non-zero OPTIMIZE level. You must, however, use the Debug Tool command SET WARNING OFF and you may get unpredictable results.</p> <p>The NOTEST option is enhanced to include the suboptions DWARF and NODWARF</p> <p>Note: Even though DWARF debugging information is always placed in the object program as NOLOAD segments, these NOLOAD segments will not take storage at runtime, unless Debug Tool, CEEDUMP, Fault Analyzer, Application Performance Analyzer or a 3rd-party vendor tool that uses DWARF debugging data is used</p>
OPTIMIZE	<p>The OPTIMIZE option is modified to allow more levels of performance optimization for your application. The previous OPTIMIZE option format is deprecated but is tolerated for compatibility.</p> <p>Note: Although OPT(0) is equivalent to the NOOPTIMIZE option in previous compilers, it now removes some code that previously was not removed.</p> <p>The storage optimization provided by the old FULL suboption of OPT is now provided by the new compiler option STGOPT</p>

For a detailed list of options supported for the various compiler versions, see Appendix E, “Option comparison,” on page 263.

For detailed descriptions of all options, see the *Enterprise COBOL Programming Guide*.

Changes in compiling with Enterprise COBOL Version 5.1

There are a number of changes to IBM Enterprise COBOL for z/OS that result in different behaviors.

The COBOL runtime library, the Language Environment component of z/OS, must now be available at compilation time. In addition, Language Environment must be updated with the APAR fixes (PTFs) for compiling programs with Enterprise COBOL Version 5 and for running programs that were compiled with Enterprise

COBOL Version 5. For details about prerequisite software levels and required maintenance, see “Prerequisite software and service for Enterprise COBOL V5” on page 179.

Compile-time storage requirements are substantially increased compared to prior versions of Enterprise COBOL. The compiler requires a minimum of 200M REGION size to run. The compiler option SIZE(MAX) is no longer supported, but gets tolerated and interpreted as SIZE(5000K). Your SIZE option setting should be in the range of 5000 K to 20000 K and your region size should be at least 200M. The region size must be large especially at higher optimization levels, that is, programs compiled with the OPT(1) or OPT(2) compiler option.

Note: If you get unexpected compiler abends or this message: IEW4000I FETCH FOR MODULE IGYCBE FROM DDNAME STEPLIB FAILED BECAUSE INSUFFICIENT STORAGE WAS AVAILABLE., make sure that your region size is at least 200M. REGION=0M in JCL gives you the maximum amount allowed by the JES system defaults set up by your system programmer. It may be less than needed. In that case your system programmer must increase the user limit of region size.

It is not necessary to specify a high SIZE value for every large program. You must raise the default SIZE value only when you encounter this error message during compilation: IGYPG5062-U THERE WAS INSUFFICIENT STORAGE FOR COMPILER PROCESSING. This message indicates that the compiler front end has run out of memory while still processing the program, and you must use the SIZE option to allocate more memory for the front end.

However, note that the memory allocated to the front end using the SIZE option is not available to later phases of the compilation. Therefore, carefully calibrate the SIZE value to avoid depriving the code generation and optimization steps of memory. Otherwise, the compiler might abend in those later phases with the following message: IGYCB7145-U INSUFFICIENT MEMORY IN THE COMPILER TO CONTINUE COMPILATION.

For more information about the SIZE compiler option, see *SIZE* in the *Enterprise COBOL Programming Guide*.

Consider also the following changes:

- The Language Environment member ID for Enterprise COBOL Version 5 Release 1 is 4 (The member ID for all previous COBOL versions was 5).
- Compile-time CPU time requirements are substantially increased, compared to prior versions of Enterprise COBOL. The compiler may take more than four times as long to compile as the older compilers.
- Compile time and run time diagnostic messages might differ, and might be generated at different times or locations.
 - Presence or absence of informational and warning level diagnostic messages might differ
 - Diagnostic messages for programs that define excessive and unsupported amounts of storage might be issued either by the binder at bind time, or by Language Environment at run time, instead of by the compiler at compilation time.
- The compiler output is in GOFF format. This format allows the compiler to put out the NOLOAD debug information (DWARF) segments.
- There is no SYSDEBUG data set created for debug information.

- Compiler listing format and contents differ from prior versions of Enterprise COBOL. You can find details on these changes in the *Enterprise COBOL Programming Guide*.
- Several compiler limits are increased with Enterprise COBOL V5. For details, see Appendix F, “Compiler limit comparison,” on page 279.

Compiler output to uninitialized data sets not supported

There are a couple of cases where the compiler fails if it tries to write to uninitialized data sets.

Sequential data sets

With Enterprise COBOL Version 4 and earlier, the compiler could write to a pre-allocated object file with no specific attributes from a previous compile step. This is not possible with Enterprise COBOL Version 5.

For example, with Enterprise COBOL Version 4, the compiler could write to a pre-allocated data set with no specified attributes (DISP=MOD) from a previous step. When the compiler had written to the data set, it had the following attributes:

```
RECFM=FB LRECL=80 BLKSIZE=3200 DSORG=PS
```

With Enterprise COBOL Version 5, the attributes are not changed and the attempt to write to the file fails.

The file attributes will be

```
RECFM=U LRECL=** BLKSIZE=6144 DSORG=PS
```

This is not valid input to the binder.

To address this, you can provide data control block (DCB) information as follows on the preallocation:

```
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
```

PDS or PDSE data set

With earlier versions of Enterprise COBOL, the compiler could write to a pre-allocated PDS object file with no specific attributes from a previous compile step. This is not supported with Enterprise COBOL Version 5.

For example, with Enterprise COBOL Version 4, the compiler could write to a pre-allocated PDS or PDSE with no specified attributes (DISP=MOD) from a previous step. The compiler will create an object file of attributes:

```
RECFM=FB LRECL=80 BLKSIZE=3200 DSORG=PO
```

With Enterprise COBOL Version 5 DISP=MOD is not supported for PDS or PDSE data sets.

If the PDS has undefined format (such as output from a previous step with no DCB), and you use DISP=SHR or DISP=OLD, Enterprise COBOL Version 5 will write but will not change the attributes. They will be left as:

```
RECFM=U LRECL=** BLKSIZE=6144 DSORG=PO
```

which is not valid input to the binder.

To fix this, specify DCB information on the allocation step as:

DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)

Do not use DISP=MOD. Use only DISP=SHR or DISP=OLD.

JCL and packaging changes for Enterprise COBOL V5.1

There have been a number of changes to the packaging, installation and JCL with Enterprise COBOL V5.1.

The SIGYCOMP data set is now a PDSE, rather than a PDS data set as in prior versions.

Enterprise COBOL V5.1 requires additional data sets

- When compiling under z/OS TSO or batch, the COBOL compiler now requires 15 utility data sets, SYSUT1 to SYSUT15
- The SYSMDECK data set is now required for all compilations. SYSMDECK may be specified as a utility (temporary) data set if the NOMDECK option is specified. When MDECK is specified, the SYSMDECK DD allocation must specify a permanent data set.
- The alternate DDNAME list parameter, used when the COBOL compiler is invoked from an assembly language program, is expanded with entries for the additional work data sets.

The following JCL cataloged procedures are no longer supported, and have been deleted with Enterprise COBOL V5.1. Because they all use the Language Environment Prelinker or the DFSMS Loader, which are no longer supported for use with Enterprise COBOL V5.1.

- IGYWCG
- IGYWCPG
- IGYWCPL
- IGYWCPLG
- IGYWPL

The catalogued procedures that ship with Enterprise COBOL V5.1 have been modified.

- IGYWC
- IGYWCL
- IGYWCLG

Link edit/bind changes with Enterprise COBOL Version 5.1

There have been a number of changes to link editing or binding Enterprise COBOL V5.1 programs.

- The DFSMS Program Management Binder must be used to bind (link edit) Enterprise COBOL V5 applications. The Language Environment Prelinker is no longer supported.
- Executables are Program Objects, not LOAD MODULES. The Program Management Loader (IEWBLDGO) is no longer supported.
- Executables cannot reside in PDS (only in PDSE)

- NOLOAD segments will not take storage at run time, unless Debug Tool, CEEDUMP, Fault Analyzer, Application Performance Analyzer or a 3rd-party vendor tool that uses DWARF debugging data is used
- If a Program Object contains programs with RMODE 24, or if the binder option AMODE(24) is specified, the binder option RMODE(24) must also be specified. A program can get an RMODE of 24 in the following cases:
 - The program is Enterprise COBOL that is compiled with the RMODE(24) or the NORENT option.
 - The program is VS COBOL II that is compiled with the NORENT option.
 - The program is assembler that contains a CSECT with RMODE 24.
 - COBOL programs compiled with a compiler earlier than V5 that run with AMODE 24 and statically call a COBOL program compiled with COBOL V5 or later.

Changes at run time with Enterprise COBOL V5.1

There are a number of changes to runtime behavior with Enterprise COBOL V5.1.

If a z/OS system does not have Language Environment PTFs installed to support Enterprise COBOL Version 5.1 programs, you cannot run Enterprise COBOL Version 5.1 programs on that system.

- Runtime option changes. For details, see “Language Environment option changes” on page 188.
- Interoperability. Enterprise COBOL Version 5 has some restrictions with interoperability with older versions of COBOL. For details see, “Interoperability with older levels of IBM COBOL programs” on page 19.
- All of the AMODE and RMODE scenarios supported by Enterprise COBOL Version 4 are now supported with Version 5, except that programs compiled with the NORENT compiler option must be RMODE 24. After binding, executable COBOL programs can have any of the following combinations of AMODE and RMODE attributes:
 - AMODE 31 and RMODE ANY
 - Either AMODE ANY or AMODE 31, and RMODE 24
 - AMODE 24 and RMODE 24

The resolved AMODE and RMODE settings depend on the COBOL language constructs used, the compiler options specified, the binder options specified, and the AMODE and RMODE attributes of the input object modules that are bound into the executable module.

- For applications compiled with Enterprise COBOL V5, the compiled-in range checks cannot be disabled at run time using the runtime option CHECK(OFF) or NOSSRANGE .
- The ILBOABN0 interface for requesting an ABEND in a COBOL environment can be called dynamically with Enterprise COBOL V5 and later versions. When called by a program compiled with Enterprise COBOL compiler, it will have the same result as calling CEE3ABD using ACTION code 1.

You are strongly recommended to migrate and use the CEE3ABD interface, because the CEE3ABD interface provides extra flexibility to control the level of details provided in the CEEDUMP produced.

When your application is called by Enterprise COBOL programs, it might ABEND in an unexpected way if it has an older version of ILBOABN0 (before LE's SCEELKED) statically linked. To fix the unexpected ABEND, you can follow one of the advises below:

- Migrate to CEE3ABD.
 - Relink your application with the REPLACE ILB0ABN0 in the LINK step, against LE's SCEELKED.
 - Change the COBOL program to use dynamic call for ILB0ABN0.
 - The IGZERRE and ILB0STP0 interfaces for managing a reusable COBOL environment are not supported for applications containing programs compiled with Enterprise COBOL V5.
 - The IGZBRDGE macro, for converting static calls to dynamic calls, is not supported for programs compiled with Enterprise COBOL V5.
 - A correction has been made to the way Enterprise COBOL handles conflicts with record length in READ statements for variable-length record files. For details, see “Variable length records - wrong length READ” on page 188.
 - Invalid COBOL programs may behave differently with Enterprise COBOL V5 than with prior versions. You should consider more vigorous testing for migrating to Enterprise COBOL V5 than you did for migrating to Enterprise COBOL V4.
 - Programs that use unsupported (yet undiagnosed) COBOL language syntax.
 - Programs referencing data items that at run time contain values not conforming to the PICTURE clause in the data description entry. For example:
 - a fullword binary item with picture S9(6) USAGE BINARY, containing an oversize value of +123456789 (unless the TRUNC(BIN) option was specified)
 - a two-byte packed-decimal item with picture S99 PACKED-DECIMAL, containing an oversize value of 123 (for example, 123C in hexadecimal).
 - a packed-decimal or zoned-decimal item containing an invalid or non-preferred sign, that does not conform to the sign requirements of the data description entry .
 - Programs that set the value of an ODO object to outside of the legal range. See the following illegal case that is not diagnosable by the compiler:


```
01 X.
   02 VAR1 COMP-3 PIC 9(3).
   02 VAR2 PIC X OCCURS 0 to 1 depending on VAR1.

MOVE 128 to VAR1 MOVE ALL 'C' to VAR2
```
- Results:
- For V2, V3, V4: 128 bytes of 'C' were moved
 - For V5R1: 1 byte of 'C' and 127 bytes of junk was moved
 - Programs with undiagnosed subscript range errors (when the SSRANGE compiler option was not specified), that reference storage outside the storage allocation for the base data item.
 - Applications with low-level dependencies on specific generated code sequences, register conventions, or internal IBM control blocks may behave differently with Enterprise COBOL V5 than with prior versions. The information such as PROGRAM-ID, COMPILED TIME, and COMPILED DATE included in the initialization code of Enterprise COBOL V4 or earlier is not included in the initialization code of Enterprise COBOL V5, so the program it depends on might behave differently with Enterprise COBOL V5.
 - VSAM record areas for reentrant COBOL programs are allocated above 16 MB, by default. Programs that pass data in VSAM file records as CALL ... USING parameters to AMODE 24 subprograms may be impacted. Such programs can be

recompiled with the DATA(24) compiler option, or the Language Environment HEAP() option can be used, to ensure that the records are addressable by the AMODE 24 programs.

- CICS System Definition (CSD) file might need to be updated to include Enterprise COBOL V5.1 runtime modules. For details, see “CSD setup differences with Enterprise COBOL V5” on page 211.

Language Environment option changes

There have been a number of changes to runtime options for Enterprise COBOL Version 5 programs.

The following options have different behavior for programs compiled with Enterprise COBOL Version 5.

Table 34. Runtime option changes with Enterprise COBOL Version 5

Option	Comments
HEAP	<p>In some cases, WORKING-STORAGE space is not acquired from HEAP and therefore the HEAP (and STORAGE) option has no effect.</p> <p>WORKING-STORAGE is acquired from HEAP when the COBOL program is compiled with the RENT option and is in one of the following cases:</p> <ul style="list-style-type: none"> • Compiled with Enterprise COBOL V4.2 or earlier releases • Compiled with the DATA(24) compiler option • Running in CICS • A COBOL V5.1.1 in a program object that contains only COBOL programs (V5.1.1, V4.2 or earlier) and assembly programs. There are no Language Environment interlanguage calls within the program object and no COBOL V5.1.0 programs. • A COBOL V5 program in a program object where the main entry point is COBOL V5. In this case, the program object can contain Language Environment interlanguage calls, with COBOL statically linking with C, C++ or PL/I. All COBOL V5 programs within such program objects (even if they are not the main entry point) have their WORKING-STORAGE allocated from heap storage. <p>In cases where WORKING-STORAGE is not obtained from HEAP, storage tuning using the HEAP option no longer improves performance. In fact, acquiring a large HEAP to avoid repeated requests for HEAP storage is detrimental to COBOL V5.1 program performance in these cases.</p>
CHECK(OFF)	CHECK(OFF) no longer disables SSRANGE checking.
NOSSRANGE	NOSSRANGE no longer disables SSRANGE checking.
STORAGE	In many cases, STORAGE initial values for HEAP no longer affect WORKING-STORAGE initial values. For details about the cases, see the discussion of the HEAP option above.

Variable length records - wrong length READ

Enterprise COBOL Version 5.1 corrects READ statement processing of wrong-length records.

The COBOL 85 standard specifies the following rules as part of the processing of READ statements: “If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for the file,

the portion of the record area which is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for file-name-1, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status value of 04 is set indicating that a record length conflict has occurred.”.

This logic was correctly implemented in VS COBOL II, COBOL/370 and COBOL for MVS & VM. If you have programs compiled with these compilers without the APAR fixes listed below installed, your programs will get the status value of 04 when READ statements encounter a record length conflict, and this is the same behavior as that with Enterprise COBOL V5.1.

The I/O logic was changed in these compilers via the APARs listed below. If you have programs compiled with VS COBOL II, COBOL/370 or COBOL for MVS & VM with the APAR fixes installed, or any later compilers, your programs will not get the status value of 04 when READ statements encounter a record length conflict. As a result there may be hidden problems in your programs.

With Enterprise COBOL Version 5, you will get File Status 04 to help you when your programs READ a wrong-length record. You could have bad data in variables or get protection exceptions for accessing data beyond your records. These cases will now properly get File Status 4 to let you properly handle these error cases.

If your program performs a “wrong length read” as described here and your code checks for File Status = 0 after READ of variable-length record files, your code will now take the 'Not zero' path. You can change your code to test for FS=0, FS=04 and other values will all be a failed READ. For FS=04 you may want to add code to avoid the bad data in variables or protection exceptions mentioned above.

If you have programs compiled with one of the following compilers, you may have different results for READ statements with Enterprise COBOL Version 5:

- VS COBOL II V1.3 with PTFs for APAR PN34704 installed
- VS COBOL II V1.4 with PTFs for APAR PN38730 installed
- COBOL/370 V1.1 or V1.2 with PTFs for APAR PN36445 installed
- COBOL for OS/390 & VM, V2
- Enterprise COBOL V3 or V4

The following example shows differences you may see:

```
*****
**                                     **
**  DESCRIPTION: This testcase tests if a compiler exhibits                **
**  corrected COBOL V5 var length READ behavior.                          **
**                                     **
*****
Identification division.
Program-id. READVAR.
Environment division.
Input-output section.
File-control.
    Select File1
        Assign to ddvar1
        Access mode is sequential.
    Select File2
        Assign to ddvar1
        Access mode is sequential
        File status is fs2.
I-O-control.
```

```

Data division.
File section.
Fd  file1
   Record is varying
   Recording mode V.
01  Record1a      pic x(20).
01  Record1b      pic x(40).
Fd  file2
   Record is varying 10 to 40
   Recording mode V.
01  Record2a      pic x(25).
01  Record2b      PIC x(35).
Working-storage section.
1  fs2 pic 99.
1  fs3 pic 99.
1  Errflag pic x value "N".
Procedure division.
   Display "Starting READVAR"

*-----*
* Create file1 with 1 20-byte record and 1 40-byte record
*-----*
   Open output file1
   Move all "a" to record1a
   Write record1a
   Move all "b" to record1b
   Write record1b
   Close file1.

*-----*
* Read file2 with 25 and 35 byte records defined
*   First READ should get FS=04 because 20 byte record is
*   shorter than the smallest record description
*   Second READ should get FS=04 because 40 byte record is
*   longer than the longest record description
*-----*
   Open input file2
   Read file2
   If fs2 = 4 then
      Display " Corrected COBOL V5 behavior"
   Else
      Display " Incompatible wrong length read behavior"
   End-If
   Read file2
   If fs2 = 4 then
      Display " Corrected COBOL V5 behavior"
   Else
      Display " Incompatible wrong length read behavior"
   End-If
   Close file2.
   Goback.

```

Interoperability with older levels of IBM COBOL programs

There are some restrictions for Enterprise COBOL V5 programs to call or be called by (interoperate) with programs compiled with earlier versions of COBOL.

Enterprise COBOL V5 programs cannot interoperate with OS/VS COBOL or VS COBOL II NORES programs in a single application. A COBOL run unit (Language Environment enclave) that contains an Enterprise COBOL V5 compiled program must not contain any OS/VS COBOL or VS COBOL II NORES programs.

Note: Run units that contain only COBOL programs compiled with Enterprise COBOL V4 or earlier versions can interoperate with OS/VS COBOL and VS COBOL II NORES programs.

Programs compiled with Enterprise COBOL V5 can interoperate with programs compiled with VS COBOL II or later, based on the following conditions and CALL types:

- Static calls. Enterprise COBOL V5 compiled programs can be bound or link-edited with the following object modules or programs to form a single program object. The programs within the program object can specify static calls to and from each other.
 - Programs that are compiled with VS COBOL II with the RES compiler option
 - Programs that are compiled with any IBM COBOL compiler versions subsequent to VS COBOL II
 - Programs that are compiled with Enterprise COBOL V3 or V4

Note: Programs that are compiled with VS COBOL II with the NORES compiler option specified cannot interoperate with programs compiled with Enterprise COBOL V5.

- Dynamic calls. Program modules that contain programs compiled with VS COBOL II with the RES option, or subsequent versions of COBOL can also interoperate with Enterprise COBOL V5 program objects by using dynamic CALL statements.
- DLL calls. Program modules that are compiled with earlier versions of COBOL that supported DLL linkage can interoperate with Enterprise COBOL V5 program objects by using DLL linkage.

Debug information changes with IBM Enterprise COBOL Version 5

Programs compiled with IBM Enterprise COBOL Version 5 will have different debug information than that of programs compiled with previous versions of the compiler.

IBM Enterprise COBOL Version 5 solves the dilemma of debugging information. In the past you had 2 choices:

- Have the debug data always with the executable at a cost of a large load footprint, or
- Have separate debug data but also have the challenge of keeping it synchronized with the application and finding it when needed.

Now you have the best of both worlds. With NOLOAD debug segments in the program object, the debug data does not increase the size of the loaded program, it always matches the executable and is always available so there is no need to search lists of data sets.

There have been changes to the TEST compiler option used to generate debuggable versions of your application and to the NOTEST option.

- When the TEST option is specified, DWARF debug information is included in the application module.
- If the SOURCE suboption is specified, the DWARF debug information includes the expanded source code, and the compiler listing is not needed by IBM Debug Tool. When the TEST(NOSOURCE) compiler option is specified, the generated DWARF debugging information does not include the expanded source code.
- You can use the NOTEST(DWARF) compiler option to include basic DWARF debugging information in the program object. You cannot debug such programs with Debug Tool, but you can get NOTEST optimization and still enable application failure analysis tools, such as CEEDUMP output and IBM Fault Analyzer.

- To have no debugging information in the program object, use the NOTEST(NODWARF) option.

When debugging your COBOL programs, you will find that there have been a large number of improvements and behavior changes introduced with Enterprise COBOL V5.1. For details about changes in debugging with IBM Debug Tool, see “Debug Tool changes with IBM Enterprise COBOL Version 5” on page 204.

WORKING-STORAGE SECTION changes

You can use the following method to locate the WORKING-STORAGE in Enterprise COBOL V5 programs at run time.

To find the start of WORKING-STORAGE in COBOL V5, you need to know how to locate the PPA4 (Program Prologue Area 4) in a dump.

How to find the PPA4 (Program Prolog Area 4) in a dump?

1. Find the start of the program in the dump from the traceback.
2. At the starting address + x'0C' is an offset value. This is the offset to the PPA1 from the start of the program.
3. Starting address + PPA1 offset = PPA1.
4. Go there in the dump.
5. At PPA1 + x'04' is an offset value. This is the offset to the PPA2 from the start of the program.
6. Starting address + PPA2 offset = PPA2.
7. Go there in the dump.
8. At PPA2 + x'08' is an offset value. This is the offset to the PPA4 from the PPA2 address.
9. PPA2 + PPA4 offset = PPA4.
10. Go there in the dump. You are now at the PPA4.

Next, you need to know the layout of the PPA4.

PPA4 layout

The major fields in PPA4 that you need are as follows:

Offset	Length	Description	
X'08'	4	Address of NORENT static	A(NORENTstatic)
X'0C'	4	Signed offset from WSA to 32-bit RENT static	Q(RENTstatic)
X'10'	4	Signed offset from 32-bit RENT static to program static address cell. Note: You need to dereference (get the value in storage at) the address cell to get the address of the program static area.	A(DATA24_31_address_cell-RENTstatic)
X'14'	4	Offset of user code from PPA4	A(Code-PPA4)
X'18'	4	Length of user generated code	Code Length
X'1C'	4	Length of NORENTstatic area	Length NORENTstatic
X'20'	4	Length of RENTstatic area	Length RENTstatic
X'24'	4	Length of DATA31 area	Length DATA24_31
X'28'	4	Offset of program name from PPA4	A(CUName-PPA4)
X'2C'	4	Offset of WORKING-STORAGE (from Q(RENTstatic) or A(DATA24_31_address_cell-RENTstatic))	
X'30'	4	Length of WORKING-STORAGE	
X'34'	1	Byte with bit to indicate if the program has EXTERNAL data items	

For information about each PPA4 offset, length, and description, see the COBOL V5 32-bit PPA4 layout table in the *z/OS Language Environment Vendor Interfaces*.

Next, you need to know some terminology.

Terms to know

NORENT static area

This storage area is allocated in the executable for each program that was compiled with NORENT. A NORENT program's WORKING-STORAGE will be located here.

LE's writable static area (WSA)

Every COBOL V5 program object (executable) has this storage area.

RENT static area

This storage area is allocated inside the WSA for every program that is statically bound into the executable and compiled with RENT. Each program has their own RENT static area. A program's WORKING-STORAGE may or may not be located here.

Program static area

This storage area is allocated outside of the WSA only if certain conditions are met. In those cases, the program's WORKING-STORAGE will be located here, instead of in the RENT static area.

Next, you need to understand that there are three locations where WORKING-STORAGE can reside.

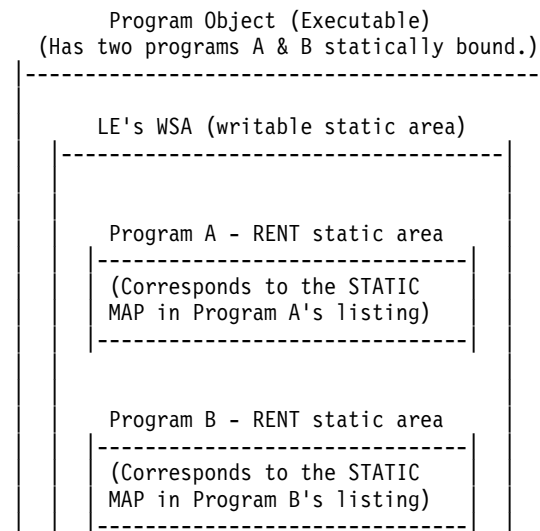
Explanation of the areas where WORKING-STORAGE can reside

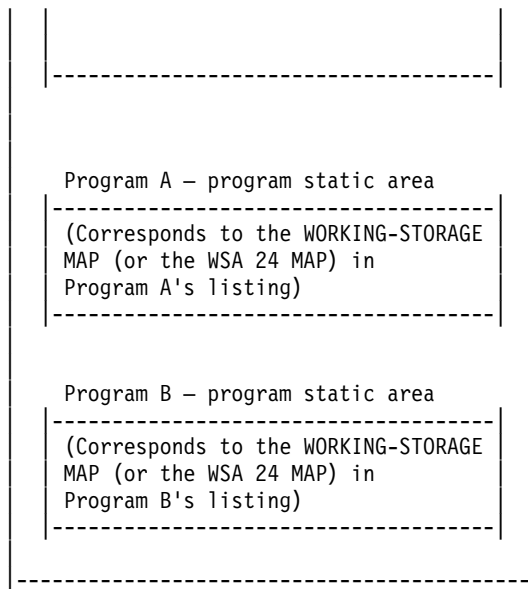
There are three different locations where WORKING-STORAGE can reside:

- Inside the program object (executable). All programs compiled with the NORENT option have a NORENT static area reserved within the executable and WORKING-STORAGE resides here.
- All programs compiled with the RENT option have a RENT static area allocated inside LE's WSA (writable static area). WORKING-STORAGE could reside here.
- Instead of being located in the RENT static area, some COBOL V5 or later RENT programs have their WORKING-STORAGE allocated outside of LE's WSA, in an area called the program static area.

The rules for determining where WORKING-STORAGE resides are located in the next section.

The picture below shows how storage is laid out for RENT programs whose WORKING-STORAGE resides in the program static area:





Once you understand the three areas where WORKING-STORAGE could reside, you need to know how to determine where a program's WORKING-STORAGE actually does reside.

How to determine the area where WORKING-STORAGE is located?

Table 35. Area where WORKING-STORAGE is located

COBOL versions	Compiler options	Where is WORKING-STORAGE located?
COBOL V5	NORENT	In the program's NORENT static area
	RENT, DATA(31)	In the program's RENT static area inside the WSA
	RENT, DATA(24) or RENT, WSOPT	In the program's program static area outside the WSA

Once you know what area the WORKING-STORAGE resides in, then you will know how to find it.

How to find WORKING-STORAGE in a dump?

Table 36. How to find the PPA4, NORENT static area, LE's WSA, RENT static area, and program static area in a dump?

What to find?	How to find it in a dump?
PPA4	See the instructions above.
NORENT static area	The address is located in storage at <PPA4 + x'08'>
LE's WSA	The address is located in storage at <CEECAA (or R12) + x'1F4'>. This is called CEECAARENT in a dump.

Table 36. How to find the PPA4, NORENT static area, LE's WSA, RENT static area, and program static area in a dump? (continued)

What to find?	How to find it in a dump?
RENT static area	The address is located in storage at <The address in storage at CEECAA (or R12) + x'1F4'> + <the offset in the program's PPA4 + x'0C'>
Program static area	The address is located in storage at <The address in storage at CEECAA (or R12) + x'1F4'> + <the offset in the program's PPA4 + x'0C'> + <the offset in the program's PPA4 + x'10'>

Once you find these areas in a dump, then you can compare that to the compile listing.

In a COBOL listing:

- The STATIC MAP shows the layout of the RENT static area or the NORENT static area.
- The WORKING-STORAGE MAP or the WSA 24 MAP shows the layout of the program static area.

related tasks

Reading LIST output (Enterprise COBOL Programming Guide)

related references

Example: Program prolog areas (Enterprise COBOL Programming Guide)

Common interfaces and conventions (z/OS Language Environment Vendor Interfaces)

Chapter 16. Adding Enterprise COBOL V5.1 programs to existing COBOL applications

When you add an Enterprise COBOL program to an existing application, you are either recompiling an existing program with Enterprise COBOL or including a newly written Enterprise COBOL program.

Note: You should use this Migration Guide only if you have completed the runtime migration to Language Environment. This means that the following conditions have been met:

- The Language Environment dataset SCEERUN is installed in LNKLIST or LPALST
- There are no instances of COBLIB, VSCLLIB or COB2LIB in LNKLIST or LPALST
- There are no instances of COBLIB, VSCLLIB or COB2LIB in JCL STEPLIB or JOBLIB statements or in CICS startup JCL
- All statically bound runtime library routines for programs that are compiled with NORES have been REPLACed with routines from Language Environment.
- IGZEBST bootstrap modules for VS COBOL II programs that are compiled with RES were either linked with the VS COBOL II runtime version of IGZEBST that has APAR PN74000 applied, or IGZEBST was REPLACed with IGZEBST from Language Environment.

If these steps have not been completed, please first complete all runtime migration activities in the *Enterprise COBOL Version 4.2 Compiler and Runtime Migration Guide* prior to following the steps here.

When you add Enterprise COBOL programs to your existing applications, you have the ability to:

- Upgrade your existing programs incrementally, as your shop's needs dictate
- Use Language Environment condition handling

If you have a program object that includes a COBOL program linked with C, C++, or Enterprise PL/I programs, the program object has slightly different behavior when the COBOL program is changed to Enterprise COBOL V5. This occurs when such program objects are fetched (that is, using either C fetch or PL/I fetch) more than once. In the subsequent fetches, external and static variables in these other LE languages may retain their last used state, following COBOL rules, instead of getting their initial values. With prior versions of COBOL linked in, the C, C++ and PL/I programs would retain C/C++ or PL/I behavior.

Link-editing restriction:

You cannot mix Enterprise COBOL V5.1 programs with:

- OS/VS COBOL programs. You must migrate to Enterprise COBOL. To find any OS/VS COBOL programs you can:
 - use the LMA tool of Debug Tool to scan load libraries for OS/VS COBOL programs
 - use the Edge Portfolio Analyzer to scan load libraries for OS/VS COBOL programs

- install the fix for APAR PM86742 to your Language Environment and look for a Warning message about detected OS/VS COBOL programs at run time
- VS COBOL II NORES programs. You must migrate to Enterprise COBOL.

AMODE restrictions with Enterprise COBOL Version 5 programs:

AMODE 24 execution is not supported in the following cases, and the applications must run in AMODE 31:

- Programs containing XML PARSE statements
- Programs containing XML GENERATE statements
- Program objects containing COBOL bound together with C, C++, or PL/I programs, and communicating via static CALL
- Programs containing the object-oriented language syntax, such as INVOKE statements, or object-oriented class definitions
- Programs compiled with any of the following compiler options:
 - DLL
 - PGMNAME(LONGUPPER)
 - PGMNAME(LONGMIXED)
- Multithreaded applications

Note: A program compiled with the THREAD option can run in AMODE 24, but only in an application that does not have multiple threads or PL/I tasks.

- Programs run from the z/OS UNIX file system

Note: An AMODE 31 driver program resident in the z/OS UNIX file system can contain a dynamic call to an AMODE 24 program module resident in an MVS PDS or PDSE.

- Programs used as COBOL compiler exit modules that are specified on the EXIT compiler option
- Language Environment enclaves that use XPLINK, including either the enclaves that contain non-COBOL programs compiled with the XPLINK compiler option, or run with the XPLINK runtime option

Note: To run COBOL programs with AMODE 24, you must compile all COBOL programs with Enterprise COBOL V5.1.1 or later versions; or Enterprise COBOL V4.2 or earlier versions. If any component of a program object is compiled with Enterprise COBOL V5.1.0, the program object must run in AMODE 31. COBOL programs that run with AMODE 24 must be linked with the binder option RMODE(24).

The ILB0ABN0 interface for requesting an ABEND in a COBOL environment can be called dynamically with Enterprise COBOL V5 and later versions. When called by a program compiled with Enterprise COBOL compiler, it will have the same result as calling CEE3ABD using ACTION code 1.

You are strongly recommended to migrate and use the CEE3ABD interface, because the CEE3ABD interface provides extra flexibility to control the level of details provided in the CEEDUMP produced.

When your application is called by Enterprise COBOL programs, it might ABEND in an unexpected way if it has an older version of ILB0ABN0 (before LE's SCEELKED) statically linked. To fix the unexpected ABEND, you can follow one of the advises below:

- Migrate to CEE3ABD.
- Relink your application with the REPLACE ILB0ABN0 in the LINK step, against LE's SCEELKED.
- Change the COBOL program to use dynamic call for ILB0ABN0.

RMODE restrictions with Enterprise COBOL Version 5 programs:

- Reentrant programs may be RMODE 24 or RMODE ANY
- Non-reentrant programs must be RMODE 24.

Enterprise COBOL V5 programs continue to support dynamic CALL to or from AMODE 24 programs. For example:

- Enterprise COBOL V5 programs may dynamically CALL AMODE 24 programs created by prior versions of Enterprise COBOL.
- AMODE 24 COBOL programs created by prior versions of Enterprise COBOL may dynamically CALL Enterprise COBOL V5 programs.
- Enterprise COBOL V5 programs may dynamically CALL AMODE 24 assembler language programs.

AMODE and RMODE considerations

Static calls between AMODE 24 and AMODE 31 programs are not supported by Enterprise COBOL V5.1.0 programs. Static calls between AMODE 24 programs and Enterprise COBOL V5.1.1 programs are supported for the cases where AMODE 24 is supported for Enterprise COBOL V5.1.1 programs. In addition, NORENT programs can no longer reside above the line. The following diagram shows the types of calls that can be dynamic or static and those that can only be dynamic. It also shows configurations of data and program location with respect to the 16 MB line.

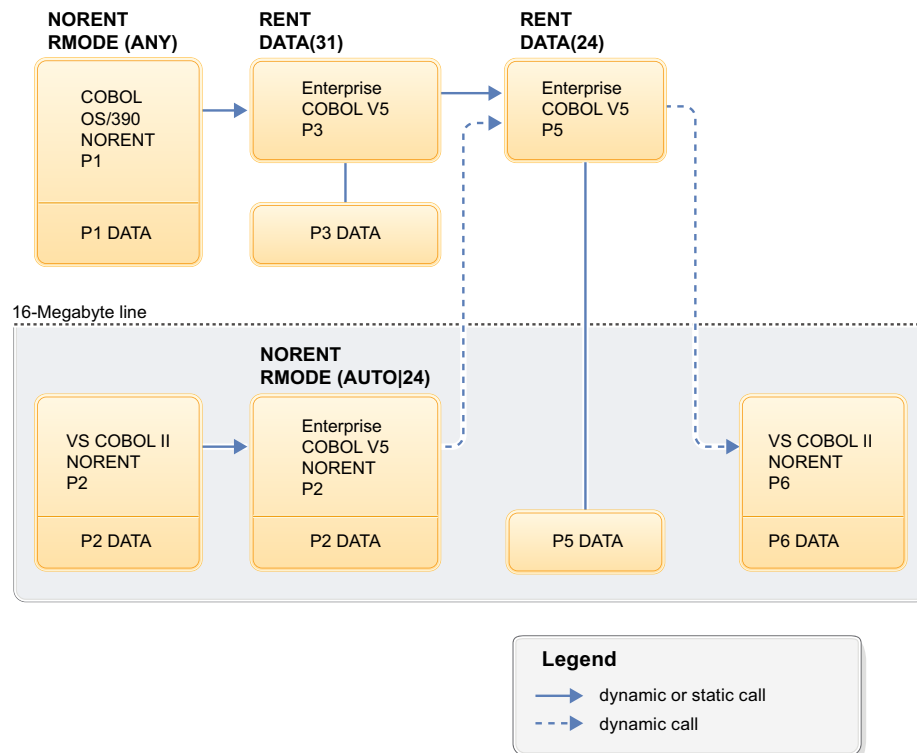


Figure 5. Valid dynamic and static calls between different AMODE and RMODE COBOL programs

Note: For other AMODE 24 programs, no calls are allowed between Enterprise COBOL V5 programs and either OS/VS COBOL or VS COBOL II NORES programs.

Part 5. Enterprise COBOL migration and other IBM products

Enterprise COBOL for z/OS, V5.1 gives you access to CICS, DB2, IMS and other data and transactional systems. It can also be used with Debug Tool.

Chapter 17. Debug tool

Debug Tool is a program analyzer that runs within Language Environment and supports a number of high-level languages, including Enterprise COBOL.

Debug Tool provides support for VS COBOL II Release 3.0 and all subsequent COBOL compilers.

Initiating Debug Tool

When you use Debug Tool, the application program starts first and the Language Environment TEST runtime option controls the invocation of Debug Tool.

You can also invoke Debug Tool directly from your application by using the Language Environment callable service CEETEST. A brief description of these two methods follows.

TEST runtime option

The Language Environment TEST runtime option is used to determine if 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 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 you run the program. However, if debugging services are needed, you can invoke Debug Tool by using the library service CEETEST.

For detailed information about the Language Environment TEST option subparameters and suboptions, see the *Language Environment Programming Reference*.

CEETEST

Language Environment provides callable service CEETEST to allow Debug Tool to gain control, and to specify a string of commands to be passed to Debug Tool. Calling this service, causes Debug Tool to be initialized and invoked. (If Debug Tool is already initialized, then this re-entry is similar to a breakpoint.)

When using CEETEST to invoke Debug Tool, the string parameter containing a command list is optional. If you do use a command list, the commands are passed to 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), 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 the *Language Environment Programming Reference*.

Debug information changes with IBM Enterprise COBOL Version 5

Programs compiled with IBM Enterprise COBOL Version 5 will have different debug information than that of programs compiled with previous versions of the compiler.

IBM Enterprise COBOL Version 5 solves the dilemma of debugging information. In the past you had 2 choices:

- Have the debug data always with the executable at a cost of a large load footprint, or
- Have separate debug data but also have the challenge of keeping it synchronized with the application and finding it when needed.

Now you have the best of both worlds. With NOLOAD debug segments in the program object, the debug data does not increase the size of the loaded program, it always matches the executable and is always available so there is no need to search lists of data sets.

There have been changes to the TEST compiler option used to generate debuggable versions of your application and to the NOTEST option.

- When the TEST option is specified, DWARF debug information is included in the application module.
- If the SOURCE suboption is specified, the DWARF debug information includes the expanded source code, and the compiler listing is not needed by IBM Debug Tool. When the TEST(NOSOURCE) compiler option is specified, the generated DWARF debugging information does not include the expanded source code.
- You can use the NOTEST(DWARF) compiler option to include basic DWARF debugging information in the program object. You cannot debug such programs with Debug Tool, but you can get NOTEST optimization and still enable application failure analysis tools, such as CEEDUMP output and IBM Fault Analyzer.
- To have no debugging information in the program object, use the NOTEST(NODWARF) option.

When debugging your COBOL programs, you will find that there have been a large number of improvements and behavior changes introduced with Enterprise COBOL V5.1. For details about changes in debugging with IBM Debug Tool, see “Debug Tool changes with IBM Enterprise COBOL Version 5.”

Debug Tool changes with IBM Enterprise COBOL Version 5

Programs compiled with IBM Enterprise COBOL Version 5 will have many debugging advantages over programs compiled with previous versions of COBOL when debugged with Debug Tool.

For details about Debug Tool interfaces with COBOL applications, see the documentation available at: <http://www-01.ibm.com/software/awdtools/debugtool/library/>.

Most of these differences apply to all debugging modes: full screen, batch, and remote. Complete details of Debug Tool commands are described in *Debug Tool References and Messages*.

DESCRIBE ATTRIBUTES commands

The PIC string shown in Debug Tool appears as it is specified in the source and not normalized as it was prior to Enterprise COBOL Version 5.

Level members are shown as written in the source code and not normalized as they were prior to Enterprise COBOL Version 5.

There are clearer data descriptions. For example, you could now see:

```
S9(5) SIGN LEAD SEP DISP
```

instead of

```
S9(5) DSLS
```

DESCRIBE ATTRIBUTES shows the length and typed of symbolic characters with Enterprise COBOL Version 5. With prior versions of the compiler, only zeros were shown.

For condition names (level 88) , an address of 000000000 is no longer shown.

There is more compact and clearer output for an array and array element. For example:

- INDEX is displayed for type instead of IX
- The level 00 is not displayed
- There is no repetition of the type for each array element, the element type is shown only once.

Debug Tool no longer displays an address for DESCRIBE ATTRIBUTES of a register, such as %GPR0, because registers do not have addresses.

LIST command and AUTOMON output

LIST or AUTOMON of tables always shows the new Debug Tool, V12.1 option SET LIST BY SUBSCRIPT format.

When listing a record or group that contains a zero length ODO table, any data items that follow that table within the record or group are displayed. Previously, they were not.

No message is displayed for program entry when AUTOMONITOR is active.

Debug Tool variables of category Alphanumeric will be displayed within single quotation marks. For example, if you execute LIST %SYSTEM, you will now see %SYSTEM = 'MVS'.

The output of LIST %HEX has improved. The output of LIST %HEX(var) no longer shows %HEX in the output. Now the output is SBIN0_5 = X'00003039' instead of %HEX (SBIN0_5) = X'00003039'. The X' indicates a hex representation.

The output of LIST varname no longer includes the block qualification. For example, the result could be varname = 5 instead of block_name ::>varname = 5.

The output of the LIST NAMES command now displays 01 and 77 level data items. In previous versions, all data items, including subordinate data items within a record or group hierarchy were shown. To see the entire expanded structure, use DESCRIBE ATTRIBUTES varname.

LIST NAMES LABEL now only displays labels in active blocks in nested programs. Previously, all labels for the program were displayed regardless of which block you were in.

LIST TITLED output for nested programs is modified. Now only variables in active blocks are displayed.

The formatted display of an array after the LIST command has changed for COBOL. When the elements of an array are groups, all members of that group are listed together for a given element, followed by the members of the group for the following element, and so on. Previously, a given member would be listed across all array elements, and then the next member of the group would be listed across all array elements. The keyword SUB is no longer displayed.

AUTOMONITOR output shows ADDRESS OF *var* and LENGTH OF *var* as single references.

AT APPEARANCE and LIST NAMES CUS has changed. Debug Tool is aware of cus. For example, if the main load module in your application is MYMAIN, the main program is MYMAIN, and the second program in the load module is MYSUB1, you can stop at MYMAIN:;>MYMAIN 1, you will see the following new behaviors:

- When you issue LIST NAMES CUS, the display shows the Load Module MYMAIN, and both the main program MYMAIN and the subprogram MYSUB1.
- When you issue an AT APPEARANCE breakpoint for MYSUB1, the breakpoint is accepted.

Enterprise COBOL V5 assigns a save area for each nested program. You can see these save areas with commands, such as LIST CALL.

MOVE, COMPUTE, IF commands

The MOVE and COMPUTE commands in Debug Tool have expanded to allow the same data types as the compiler for receivers and senders. This enhancement removes previous restrictions on the use of those commands.

The IF command has been expanded. Allowable comparisons for relational conditions are expanded in Debug Tool with Enterprise COBOL V5. The allowable comparison for relational conditions (involving data items, literals, and figurative constants) are implemented according to the Enterprise COBOL Language Reference.

Index changes also improve the use of these commands:

- There is relative subscripting of index names with Enterprise COBOL Version 5.
- To conform to COBOL language rules, you can no longer index an array with index data items.
- To conform to COBOL language rules, you can no longer use IN or OF qualifiers for an index name.

STEP command

You can STEP and set breakpoints for the WHEN phrase of EVALUATE.

STEP OVER with PERFORM is now supported.

Support for COBOL types

Debug Tool now supports the correct maximum value in all binary data types. For example, an 8-byte, unsigned COMP-5 data item can contain a maximum value of 18,446,744,073,709,551,615, which is 20 digits.

INDEX (IX) and Arrays

With Enterprise COBOL V5, you cannot use a data item of type INDEX as a subscript. For example, if you have defined a data item as 77 IXDI1 USAGE IS INDEX, you cannot execute LIST ARR(IXDI1).

Index names are in the debug information in the same way as top-level (01 or 77) data items, although index names do not have level numbers. In earlier versions of Enterprise COBOL, index names are shown in the debug information along with table elements, like children of the array to which they belong. In Enterprise COBOL V5 index names are not shown when table information is listed, they are only shown when listed explicitly by name. This change is reflected in the output from the following commands:

- LIST NAMES
- LIST TITLED
- DESCRIBE ATTRIBUTES (with no argument)

With Enterprise COBOL V5, you cannot qualify an INDEX name using the name of the array to which it belongs. You also can no longer qualify a containing group or record name, as if it were a subordinate data item. For example IX3 of REC1. This was possible with earlier versions of Enterprise COBOL.

Enterprise COBOL V5 supports an increment (+) or decrement (-) operator as part of the INDEX of an array. Enterprise COBOL V4 did not support this.

With Enterprise COBOL V5 programs Debug Tool defaults to 1 if you do not specify the index of an array. With previous versions, Debug Tool listed all members of the array. If the array is declared as shown below, and you issue LIST X, Debug Tool only displays the first element in the array ARR(1) as LIST X(1). LIST ARR(n) will show X and Y for the specified index, and LIST ARR will show X and Y for all members.

```
05 ARR OCCURS 10
10 X PIC 99
10 Y PIC 99
```

For previous versions of Enterprise COBOL, when you list a single element of an array, the format of the output is as if it is an array of size 1. For Enterprise COBOL V5, the output is the same as a variable of the given type, not as an array of size 1.

Other changes

The AT CALL entry name is not supported for Enterprise COBOL V5.

Several changes are implemented for the DESCRIBE CUS command. These changes are:

- New compiler name: IBM COBOL 5.1.1
- Time Stamp is displayed: * Compiler: IBM COBOL 5.1.1 2012/01/27 13:08
- There have been many changes to compiler options.
- The type of linkage is displayed: * Its linkage is Language Environment FastLink. This is the default linkage for the compiler.

Line numbering with the NUM option and sequence of programs is different with Enterprise COBOL V5. In prior versions, a batch compile (sequence of programs in a single source) with NUM and NOLIB the line numbers start over in the second program. With Enterprise COBOL V5 the NOLIB option has been removed. The compiler behaves as though LIB is always enabled and therefore the second program in a sequence has line numbers that continue from those of the first program.

Display of National data items will include N with Enterprise COBOL V5. For example: listing of 01 nat pic N(5) value "abcde" national is NAT= N'abcde' V4: NAT = 'abcde'.

The number of digits displayed in arithmetic expressions is different with Enterprise COBOL V5. The number of digits resulting from arithmetic operations are defined in the Enterprise COBOL Programming Guide.

Sign is handled differently with Enterprise COBOL V5. The result of an arithmetic expression will have a sign if either operands are signed. In earlier versions, the sign of the result depended on the answer. The exception is the case of results from subtraction and unary minus which are always signed to guarantee correctness of the result.

Full Screen Mode changes with IBM Enterprise COBOL V5.1

These changes apply to the Full Screen Mode commands and functions.

The following commands are different between Enterprise COBOL V5 programs and those of previous compilers:

- PANEL LISTINGS and PANEL SOURCES. Both commands show the program name.
- SET DEFAULT LISTINGS. The source listing information is embedded in the object for COBOL V5 programs.
- SET DYNDEBUG OFF. COBOL V5 compiler does not support compiled-in hooks. You must have SET DYNDEBUG ON if you want to step or set breakpoints in a COBOL V5 program.
- SET LIST BY SUBSCRIPT. With COBOL V5 programs, Debug Tool displays arrays as if LIST BY SUBSCRIPT ON is always enabled. With Enterprise COBOL V4 programs, the default display on an array was SET LIST BY SUBSCRIPT OFF.
- SET PROGRAMMING LANGUAGE. The programming language for COBOL V5 is COBOL.
- SET SOURCE. The source listing information is embedded in the object. An error message is displayed when you issue this command for a COBOL V5 program.

Debug Tool changes for remote mode with IBM Enterprise COBOL V5.1

This section lists changes that apply to the remote debugger interfaces.

The changes are:

- With Enterprise COBOL V5, nodes in the tree of a monitored expression show the level number, for example, 05 VAR1. With Enterprise COBOL V4, it showed VAR1.
- With Enterprise COBOL V5, PIC is shown as part of the type information, for example, 05 SBIN1 PIC 99 COMP.
- With Enterprise COBOL V4, array type was shown as ARRAY. With Enterprise COBOL V5, it is shown by using appropriate COBOL terminology such as, 9 COMP OCCURS 2. This matches the behavior of batch/Full Screen Mode.
- With Enterprise COBOL V5, record types are shown as known to the language. For example, ALPHANUMERIC GROUP or NATIONAL GROUP. With Enterprise COBOL V4, record types were shown as CHARACTER, STRUCT, or ARRAY
- With programs compiled by Enterprise COBOL V5, array subscripts can be separated by a semicolon. This was not allowed for programs compiled with Enterprise COBOL V4 and is not allowed in full screen mode.
- With programs compiled by Enterprise COBOL V5, nested programs will now show in the Debug View.
- COBOL language provides a DECLARATIVES section to handle exceptional conditions. With Enterprise COBOL V5, when a DECLARATIVES section gets control in a Debug Tool session, the debug view shows a separate frame for it.

|
|
|

Chapter 18. CICS conversion considerations for COBOL source

To run programs under CICS, you need to be familiar with the required compiler options. You also need to take migration steps to run Enterprise COBOL programs under CICS, or to upgrade programs to use the integrated CICS translator.

Consider the following topics:

- “CSD setup differences with Enterprise COBOL V5”
- “DFHRPL setup differences with Enterprise COBOL V5” on page 212
- “Compiler options for programs that run under CICS” on page 213
- “Migrating from the separate CICS translator to the integrated translator” on page 214

OS/VS COBOL restriction

OS/VS COBOL programs no longer run under CICS. Any OS/VS COBOL programs to be run under CICS must be upgraded to Enterprise COBOL.

CSD setup differences with Enterprise COBOL V5

With the following CICS TS versions, CICS uses *system autoinstall* to install the required Enterprise COBOL V5 runtime modules so you do not need to update the CICS System Definition (CSD) file:

- CICS TS V5.4 and later
- CICS TS V5.3 with the PTF for APAR PI60389 applied
- CICS TS V5.1 and V5.2 with PTFs for APARs PI60388 and PI73184 applied

Without those PTFs applied or for earlier CICS TS versions, you must update the CSD file to include Enterprise COBOL V5 runtime modules as follows.

The normal procedure for setting up CICS involves updating the CICS System Definition file to define program modules that will be used under CICS. New library modules must be added for Enterprise COBOL V5.1. These modules are contained in the Language Environment data set SCEERUN:

- CEEEV004
- IGZXLPA
- IGZXD24
- IGZXDMR
- IGZLLIBV
- IGZXLPAK
- IGZXLPIO
- IGZXAPI
- IEWBND
- IEWBIND
- CDAEED

The member CEECCSD in the Language Environment SCEESAMP data set provides an example of this definition file. You can also add the following lines to your existing CSD file:

```
DEFINE PROGRAM(CEEEV004) GROUP(CEE)
DEFINE PROGRAM(IGZXLPA) GROUP(CEE)
DEFINE PROGRAM(IGZXD24) GROUP(CEE)
DEFINE PROGRAM(IGZXDMR) GROUP(CEE)
DEFINE PROGRAM(IGZLLIBV) GROUP(CEE)
DEFINE PROGRAM(IGZXLPA) GROUP(CEE)
DEFINE PROGRAM(IGZXLPI0) GROUP(CEE)
DEFINE PROGRAM(IGZXAPI) GROUP(CEE)
DEFINE PROGRAM(IEWBNDD) GROUP(CEE)
DEFINE PROGRAM(IEWBIND) GROUP(CEE)
DEFINE PROGRAM(CDAEEDE) GROUP(CEE)
```

DFHRPL setup differences with Enterprise COBOL V5

To run programs under CICS, consider the DFHRPL setup differences.

You must update the JCL that starts CICS. Include the hlq.SEQAMOD data set of Debug Tool, and the Language Environment runtime libraries (SCEECICS, SCEERUN, and if required by your applications, SCEERUN2) in the DFHRPL concatenation.

If you are running Enterprise COBOL V5.1 (or later) programs compiled with the TEST compiler option on CICS, you must also add system libraries MIGLIB and SIEAMIGE in the DFHRPL DD concatenation. The DFHRPL concatenation is in the CICS region startup JCL.

Compiler options for programs that run under CICS

Table 37 lists the compiler options for Enterprise COBOL programs that run under CICS.

Table 37. Compiler options for programs that run under CICS

Compiler options	Comments
CICS	<p>The CICS compiler option enables the integrated CICS translator capability. The CICS option must be specified if the source program contains CICS statements and has not been processed by the separate CICS translator.</p> <p>The CICS option requires that the NODYNAM and RENT options also are in effect. Enterprise COBOL forces on these options if DYNAM, or NORENT are specified at the same level as the CICS option.</p> <p>The CICS translator option COBOL3 is recommended, although COBOL2 is still supported.</p> <p>Choose the COBOL2 option if you are retranslating old programs that require the use of temporary variables. In particular, note that the use of temporary variables might circumvent errors that would normally occur when an argument value in a program is incorrectly defined. The COBOL2 option provides declarations of temporary variables. Because of this feature, incorrect definitions of argument values might be present, but not noticeable at run time, in programs that were translated with COBOL2. Translating these programs with the COBOL3 option can reveal these errors for the first time.</p> <p>For example, suppose you coded:</p> <pre>EXEC CICS LINK PROGRAM('XXXXXX') COMMAREA(WO-COMMAREA) LENGTH('1000') END-EXEC.</pre> <p>The length is supposed to be a binary halfword but because it is enclosed in quotation marks, it is a character string. With COBOL3 the character string will be passed directly to CICS on the CALL and will result in an error. With the COBOL2 option the length will be moved to an intermediate variable and COBOL will convert it from character string to binary halfword as part of the move. To assist with migration to the newer releases of CICS, you can use the COBOL2 option to continue to circumvent errors in the programs, rather than correcting them.</p> <p>If the NOCICS option is in effect, any CICS statements found will be flagged with S-level diagnostics and discarded.</p>
DBCS	<p>The DBCS option is the default for Enterprise COBOL. It might cause problems for CICS programs if you are using the COBOL2 CICS translator option. The fix is to use the COBOL3 translator option.</p>
NODYNAM	<p>NODYNAM is required for programs translated by the CICS translator because the CICS command-level stub cannot be dynamically called.</p> <p>Note: Dynamic calls are supported under CICS by the use of the CALL identifier format of the call statement.</p>
RENT	<p>RENT is required for CICS programs. RENT causes the compiler to produce reentrant code and allows you to place the COBOL modules in the LPA (Link PackArea) or ELPA (Extended Link Pack Area) and thus shared among multiple address spaces under CICS. Also, the modules cannot be overwritten, since the LPA and ELPA are read-only storage.</p>

Table 37. Compiler options for programs that run under CICS (continued)

Compiler options	Comments
TRUNC	<p>Use TRUNC(OPT) for CICS programs that contain EXEC CICS commands, if the program uses binary data items in a way that conforms to the PICTURE and USAGE clause for them.</p> <p>Use TRUNC(BIN) if your program uses binary data items in a way that does not conform to the PICTURE and USAGE clause for them. For example, if a data item is defined as PIC S9(8) BINARY and might receive a value greater than eight digits, use TRUNC(BIN). You can also use TRUNC(OPT) and redefine specific items as COMP-5 to improve runtime performance for the whole program.</p>

Migrating from the separate CICS translator to the integrated translator

The separate CICS translator has not been updated for newer COBOL language such as floating comment delimiters. To use the latest features of the COBOL compiler, use the integrated CICS translator.

When you migrate COBOL applications to use the integrated CICS translator:

- Delete the separate translation step from the compile process.
- Change the XOPTS translator option to the CICS compiler option. The suboptions string must be delimited with quotes or apostrophes. For example, a program to be translated by the separate CICS translator might have a CBL statement like this:

```
CBL TEST(NOEPJD), XOPTS(LINKAGE,SEQ,SP)
```

For the integrated CICS translator it must be changed to this:

```
CBL TEST(NOEPJD), CICS('LINKAGE,SEQ,SP')
```
- Move all CBL/PROCESS statements to the first lines of the source program. The integrated CICS translator does not accept comment lines preceding a CBL/PROCESS statement. The source program must conform to Enterprise COBOL rules.
- Check if you have nested programs that redefine DFHCOMMAREA. The integrated translator will not generate declarations of DFHCOMMAREA or DFHEIBLK in nested programs. DFHCOMMAREA and DFHEIBLK declarations are generated in the outermost program with the GLOBAL attribute specified. COBOL programs that depend on these generated declarations within nested programs require source changes.

Integrated CICS translator

An integrated translator eliminates the separate translation step for COBOL programs that contain CICS statements.

With the integrated translator, the COBOL compiler handles both native COBOL and embedded CICS statements in the source program. When CICS statements are encountered, the compiler interfaces with the integrated CICS translator. The integrated CICS translator takes appropriate actions and then returns to the compiler indicating what native language statements to generate.

Although the separate CICS translator is still supported in Enterprise COBOL, use of the integrated CICS translator is recommended. The integrated CICS translator improves usability and offers the highest level of functionality. The benefits of using the integrated CICS translator include:

- Enhancements in interactive debugging of COBOL applications with Debug Tool. The application can be debugged at the original source level, instead of at the level of the expanded source produced by the CICS translator.
- EXEC CICS or EXEC DLI statements can reside in copybooks, eliminating the need to translate them with an external translator before compilation.
- There is no longer a need for an intermediate data set to hold the translated version (before the program has been compiled) of the source program.
- There is only one output listing instead of two.
- Using nested programs that contain EXEC CICS statements is simplified. DFHCOMMAREA and DFHEIBLK are generated in the outermost program with the GLOBAL attribute specified on the PROCEDURE DIVISION USING of nested programs.
- Nested programs that contain EXEC CICS statements can be held in separate files and included through a COPY statement.
- REPLACE statements can now affect EXEC CICS statements.
- Binary fields in CICS control blocks are generated with USAGE COMP-5 instead of BINARY. Thus, there is no longer a dependency on the setting of the TRUNC compiler option. Any setting of the TRUNC option can be used with CICS applications that use the integrated translator, subject only to the requirements of the user-written logic within the application.

Note: The CICS documentation states that the EXCI translator option is not supported for programs compiled with the integrated CICS translator, but CICS has reversed this position. You can now compile with the EXCI translator option and ignore the warning message DFH7006I.

Compiler options for the integrated CICS translator

Table 38 lists compiler options for Enterprise COBOL programs that use the integrated CICS translator.

Table 38. Key compiler options for the integrated CICS translator

Compiler option	Comments
CICS	<p>The CICS compiler option enables the integrated CICS translator capability. The CICS option must be specified if the source program contains CICS statements and has not been processed by the integrated CICS translator.</p> <p>The CICS option requires that the NODYNAM, and RENT options also are in effect. Enterprise COBOL forces on these options if DYNAM or NORENT are specified at the same level as the CICS option.</p> <p>If NOCICS option is specified, any CICS statements found in the source program will receive S-level messages and be discarded.</p>
NODYNAM	NODYNAM is required for programs translated by the CICS translator 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 LPA or ELPA and thus shared among multiple address spaces under CICS. Also, the modules cannot be overwritten, since the LPA and ELPA are read-only storage.

Chapter 19. DB2 coprocessor conversion considerations

When you upgrade programs that use the DB2 precompiler to instead use the DB2 coprocessor, you need to be aware of differences in language elements and in code-page conversions.

Consider the following topics:

- DB2 coprocessor integration
- Language elements
- Code-page conversion

Starting with DB2 Version 8 you can no longer use the DB2 precompiler for OS/VS COBOL programs. In addition, you cannot mix OS/VS COBOL with Enterprise COBOL. Therefore, if a program needs to be changed, it must be upgraded to Enterprise COBOL.

DB2 coprocessor integration

The coprocessor eliminates the need for precompilation with the DB2 precompiler in COBOL programs containing SQL statements.

The coprocessor uses the COBOL compiler to handle both native COBOL and imbedded SQL statements in the source program. When the SQL statements are encountered, the compiler interfaces with the DB2 coprocessor. The DB2 coprocessor takes appropriate actions and then returns to the compiler typically indicating what native language statements to generate.

The separate precompiler is still supported by DB2 and Enterprise COBOL, however the coprocessor approach is the preferred and recommended solution. The coprocessor approach provides improved usability and the highest level of functionality. In particular, interactive debugging of COBOL applications with Debug Tool is enhanced when the coprocessor solution is used, since the application may be debugged at the original source level, instead of at the level of the expanded source produced by the DB2 precompiler.

The benefits of a coprocessor approach include:

- Compilation of COBOL programs with a single JOB step even if the source contains EXEC SQL (and EXEC CICS) statements.
- The ability to include source code that contains EXEC SQL statements using COPY statements is available.
- Enhancements in interactive debugging of COBOL applications with Debug Tool. The application may be debugged at the original source level, instead of at the level of the expanded source produced by the separate DB2 precompiler.
- There is only one output listing instead of two.
- REPLACE statements can now affect EXEC SQL statements.
- Nested programs that contain EXEC SQL statements can be held in separate files and included through a COPY statement.

The following job stream shows an example of using the DB2 precompiler:

```

//DB2PRE JOB ...,
// NOTIFY=GTAO,MSGCLASS=A,CLASS=A,TIME=(1,0),
// REGION=200M,MSGLEVEL=(1,1)
//PC      EXEC PGM=DSNHPC,
//        PARM='HOST(COB2),QUOTE,APOSTSQL,SOURCE,XREF'
//DBRMLIB DD DSN=GTAO.DBRMLIB.DATA(COBTST),DISP=SHR
//STEPLIB DD DSN=DSN910.SDSNLOAD,DISP=SHR
//SYSCIN  DD DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
//        SPACE=(800,(500,500))
//SYSPRINT DD SYSOUT=*
//SYSTEM  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1  DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2  DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSIN   DD *

```

```

IDENTIFICATION DIVISION.
PROGRAM-ID.COBTST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    01 RES PIC X(10).
EXEC SQL
    INCLUDE SQLCA
END-EXEC.
PROCEDURE DIVISION.
EXEC SQL
    SELECT COL1 INTO :RES FROM TABLE1
END-EXEC.
GOBACK.

```

```

//COB EXEC PGM=IGYCRCTL,
//PARM=(NODYNAM,'BUF(12288)',SOURCE,NOXREF)
//STEPLIB DD DSN=IGY.V5R1M0.SIGYCOMP,DISP=SHR
//        DD DSN=CEE.SCEERUN,DISP=SHR
//        DD DSN=CEE.SCEERUN2,DISP=SHR
//SYSIN   DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//SPACE=(800,(500,500))
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT3 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT4 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT5 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT6 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT7 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT8 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT9 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT10 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT11 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT12 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT13 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT14 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT15 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSMDCK DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)

```

The following example shows the integrated SQL coprocessor:

```

//DB2INT JOB (GTAO,F342,090,M49),'Gianni Tao',
//NOTIFY=GTAO,MSGCLASS=A,CLASS=A,TIME=(1,0),
//REGION=200M,MSGLEVEL=(1,1)
//COB EXEC PGM=IGYCRCTL,
//PARM=(NODYNAM,'BUF(12288)',SOURCE,NOXREF,SQL)
//STEPLIB DD DSN=IGY.V5R1M0.SIGYCOMP,DISP=SHR
//        DD DSN=CEE.SCEERUN,DISP=SHR

```



```

//          DD DSN=CEE.SCEERUN2,DISP=SHR
//          DD DSN=DSN910.SDSNLOAD,DISP=SHR
//DBRMLIB DD DSN=GTAO.DBRMLIB.DATA(COBTST),DISP=SHR
//SYSIN DD *

IDENTIFICATION DIVISION.
PROGRAM-ID.COBTST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    01 RES PIC X(10).
EXEC SQL
    INCLUDE SQLCA
END-EXEC.
PROCEDURE DIVISION.
EXEC SQL
    SELECT COL1 INTO :RES FROM TABLE1
END-EXEC.
GOBACK.

//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//SPACE=(800,(500,500))
//SYSPRINT DD SYSOUT=* //SYSUDUMP DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT2 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT3 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT4 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT5 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT6 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT7 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT8 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT9 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT10 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT11 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT12 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT13 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT14 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSUT15 DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)
//SYSMDECK DD UNIT=SYSDA,SPACE=(800,(500,500)),,ROUND)

```

Language elements

There are some differences in the way certain aspects of SQL code are handled between the separate precompiler and the integrated coprocessor. View the following items to take into account these differences when you change to using the coprocessor.

Continuation lines

Precompiler: Requires that an EXEC SQL statement start in columns 12 through 72; continuation lines of the statement can start anywhere in columns 8 through 72.

Coprocessor: Requires that all lines of an EXEC SQL statement be coded in columns 12 through 72, including continuation lines.

Action to migrate to coprocessor: Move any continuation of EXEC SQL statements that start in columns 8 through 11 over to start in columns 12 through 72.

COPY REPLACING with SQL INCLUDE

Precompiler: An EXEC SQL INCLUDE statement can reference a copybook that contains a nested COPY . . . REPLACING statement.

Coprocessor: An EXEC SQL INCLUDE statement cannot reference a copybook that contains a nested COPY . . . REPLACING statement,

| because EXEC SQL INCLUDE is processed identically to COPY with the
| coprocessor, and nested COPY statements cannot use REPLACING. You
| might also consider using REPLACE instead of COPY REPLACING. With
| the integrated coprocessor, REPLACE will take effect even on copybooks.
| This was not the case with the separate precompiler.

Action to migrate to coprocessor: Change your code so that COPY REPLACING is only in the original COBOL source program, not in a copybook.

FOR BIT DATA host variables

Precompiler: A COBOL alphanumeric data item can be used as a host variable to hold DB2 character data that has subtype FOR BIT DATA. An explicit EXEC SQL DECLARE VARIABLE statement that declares the host variable in question as FOR BIT DATA is not required with the precompiler.

Coprocessor: A COBOL alphanumeric data item can be used as a host variable to hold DB2 character data having subtype FOR BIT DATA only if:

- You specify the NOSQLCCSID compiler option, or
- An explicit EXEC SQL DECLARE VARIABLE statement for the host variable is specified in the COBOL program. For example:

```
EXEC SQL DECLARE :HV1 VARIABLE FOR BIT DATA END-EXEC
```

If you use the DB2 DCLGEN command to generate COBOL declarations for a table, you can create the EXEC SQL DECLARE statements automatically. To do so, specify the DCLBIT(YES) option of the DCLGEN command.

Action to migrate to coprocessor:

- Use DCLGEN to add the explicit EXEC SQL DECLARE VARIABLE FOR BIT DATA statement to the data declarations for any data items that are used as bit data and not just as character data.
- Add the explicit EXEC SQL DECLARE VARIABLE FOR BIT DATA statement to the data declarations manually.
- Use the NOSQLCCSID compiler option.

Multiple definitions of a host variable

Precompiler: Does not require host variable references to be unique.

The first definition that maps to a valid DB2 data type is used.

Coprocessor: Requires that all host variables references be unique.

If a host variable reference is not unique, the coprocessor diagnoses it as a nonunique reference. You must fully qualify the host variable reference to make it unique.

Action to migrate to coprocessor: Fully qualify any host variable references for which there are multiple definitions.

Period at the end of an EXEC SQL INCLUDE statement

Precompiler: A period is not required.

If you do specify a period, the precompiler processes it as part of the statement. If you do not specify a period, the precompiler accepts the statement as if a period were specified.

Coprocessor: A period is required. (The coprocessor treats the EXEC SQL INCLUDE statement like a COPY statement.)

Example:

```

IF A = B THEN
    EXEC SQL INCLUDE somecode END-EXEC.
ELSE
    ...
END-IF

```

Note that the period does not terminate the IF statement.

Action to migrate to coprocessor: Add a period after every
EXEC SQL INCLUDE somecode END-EXEC

statement.

REPLACE and EXEC SQL statements

Precompiler: COBOL REPLACE statements and the REPLACING phrase of COPY statements act on the expanded source created from EXEC SQL statements.

Coprocessor: COBOL REPLACE statements and the REPLACING phrase of COPY statements act on the original source program including EXEC statements, which can result in different behavior in the following examples:

```

REPLACE ==ABC ==By ==XYZ ==.
01 G.
02 ABC PIC X(10).
...
EXEC SQL SELECT *INTO :G.ABC FROM TABLE1 END-EXEC

```

With the precompiler the reference to G.ABC will be displayed as ABC OF G in the expanded source and will be replaced with XYZ OF G. With the coprocessor, replacement will not occur because ABC is not delimited by separators in the original source string G.ABC.

Action to migrate to coprocessor: Change your code to either REPLACE the qualified references (for example G.ABC) as well as the unqualified references:

```

REPLACE ==ABC ==By ==XYZ ==
==G.ABC ==By ==G.XYZ ==.

```

Or change code so that qualification is not required, stop using REPLACE for such data items, or any other means to allow the COBOL programs changed by REPLACE to compile cleanly.

Source code that follows END-EXEC

Precompiler: Ignores any code that follows the END-EXEC on the same line.

Coprocessor: Processes the code that follows the END-EXEC on the same line.

Action to migrate to coprocessor: add the floating comment indicator *> after the END-EXEC phrase.

SQL-INIT-FLAG

Precompiler: If you pass host variables that might be located at different addresses when the program is called more than once, the called program must reset SQL-INIT-FLAG. Resetting this flag indicates to DB2 that storage must be initialized when the next SQL statement runs. To reset the flag, insert the statement MOVE ZERO TO SQL-INIT-FLAG in the PROCEDURE DIVISION of the called program, ahead of any executable SQL statements that use the host variables.

Coprocessor: The called program does not need to reset SQL-INIT-FLAG. An SQL-INIT-FLAG is automatically defined in the program to aid in program portability. However, statements that modify SQL-INIT-FLAG, such as MOVE ZERO TO SQL-INIT-FLAG, have no effect on the SQL processing in the program.

Action to migrate to coprocessor: Optionally remove references to SQL-INIT-FLAG, they are not used and not needed.

Code-page conversion

There are differences in the way character conversion is handled between the separate precompiler and the integrated coprocessor. View the following items to take into account these differences when you change to using the coprocessor.

Code-page coordination between COBOL and DB2 for SQL statements

Precompiler: There is no coordination. The code page for processing SQL statements is determined from DB2 external mechanisms and defaults

Coprocessor: Code-page coordination between COBOL and DB2 for SQL statements is dependant on the SQLCCSID compile option:

- SQLCCSID:
 - The COBOL CODEPAGE(ccsid) compiler option affects processing of host variables in COBOL statements and SQL statements.
 - CCSID processing is compatible with the SQL coprocessor in Enterprise COBOL V3R4.
- NOSQLCCSID:
 - The CODEPAGE(ccsid) compiler option only affects processing of COBOL statements, it is not used for processing SQL statements.
 - The code page for processing SQL statements is determined from DB2 external mechanisms and defaults.

For more information SQLCCSID and NOSQLCCSID, see the *Enterprise COBOL for z/OS Programming Guide* section "COBOL and DB2 CCSID determination".

Chapter 20. Moving IMS programs to Enterprise COBOL V5

If you use COBOL for IMS exit routines, pay attention to some restrictions with COBOL V5.

Only IMS exits that are enabled for enhanced services can reside in PDSE data sets. In particular, COBOL users commonly use two types of exits, and they are not enabled to run out of PDSE data sets:

DFSME127	The Input Message Segment Edit user exit
DFSME000	The Input Message Field Edit user exit

If you have COBOL programs that are used as these types of IMS user exits, the programs cannot be compiled with COBOL V5. The exception is when the actual exit is an assembler program in a PDS data set that LOADs and calls a COBOL V5 program in a PDSE. To handle the cases with COBOL V5 and these users exits, you have the following choices:

- If the exit routine is COBOL, do not recompile with COBOL V5, but keep using the older COBOL version.
- If the exit routine is COBOL, change to use an assembler program that LOADs COBOL V5, or an older COBOL program that does a dynamic CALL to COBOL V5 for exit logic.
- If the exit routine is assembler that loads a COBOL program, recompile the COBOL program with COBOL V5, bind into a PDSE data set, and add that new data set to the concatenation.

IMS is in the process of enabling user exits for enhanced services, which allows them to be run out of PDSE data sets. See the list of the user exit types that are enabled for the new services in IMS V11:

ICQSEVNT(new)	The IMS CQS Event user exit
ICQSSEVT(new)	The IMS CQS Structure Event user exit
INITTERM(new)	The Initialization / Termination user exit
RESTART(new in IMS 10)	The Restart user exit
PPUE (DSFSPPUE0)	The Partner Product user exit

No additional exits were enabled in IMS 12.

The following user exit types are enabled in IMS 13:

BSEX (DFSBSSEX0)	The Build Security Environment user exit
LOGEDIT (DFSFLGE0)	The Log Edit user exit
LOGWRT (DFSFLGX0)	The Logger user exit
NDMX (DFSNDMX0)	The Non-Discardable Message user exit
OTMAIOED (DFSIOE0)	The OTMA Input / Output Exit user exit
OTMARTUX (DFSRTUX)	The OTMA Resume TPIPE Security user exit
OTMAYPRX (DFSYPX0)	The OTMA Destination Resolution user exit
RASE (DFSRSAS00)	The Resource Access Security user exit

Compiling and linking for running under IMS

For best performance in the IMS environment, use the RENT compiler option. It causes COBOL to generate reentrant code. You can then run your application programs in either preloaded mode (the programs are always in storage) or nonpreload mode, without having to recompile with different options.

IMS allows COBOL programs to be preloaded. This preloading can boost performance because subsequent requests for the program can be handled faster when the program is already in storage (rather than being fetched from a library each time it is needed).

You must use the RENT compiler option to compile a program that is to be run preloaded or as both preloaded and nonpreloaded. When you preload a load module that contains COBOL programs, all of the COBOL programs in that load module must be compiled with the RENT option.

In an application with any mixture of Enterprise COBOL, IBM COBOL, and VS COBOL II programs, the following compiler options are recommended:

Table 39. Recommended compiler options for applications with mixed COBOL programs

Enterprise COBOL	IBM COBOL	VS COBOL II
RENT	RENT	RENT and RES

You can place programs compiled with the RENT option in the LPA or ELPA. There they can be shared among the IMS dependent regions.

To run above the 16-MB line, your application program must be compiled with RENT and RMODE(ANY).

With IMS, the data for IMS application programs can reside above the 16-MB line, and you can use DATA(31) and RENT for programs that use IMS services.

The recommended link-edit attributes for proper execution of COBOL programs under IMS are as follows:

- Link as RENT load modules that contain only COBOL programs compiled with the RENT compiler option.
- To link load modules that contain a mixture of COBOL RENT programs and other programs, use the link-edit attributes recommended for the other programs.

Part 6. Appendixes

Appendix A. Commonly asked questions and answers

This section provides answers to some of the most common questions about upgrading to Enterprise COBOL and Language Environment. The questions are grouped into the following categories:

- Compatibility
- Link-editing with Language Environment
- Compiling with Enterprise COBOL
- Language Environment services
- Language Environment runtime options
- Subsystems
- z/OS
- Performance
- Service
- Object-oriented syntax, and Java 5 or Java 6 SDKs

Compatibility

Can OS/VS COBOL and VS COBOL II programs call Enterprise COBOL programs?

On non-CICS, calls between OS/VS COBOL and Enterprise COBOL are not supported. On CICS, OS/VS COBOL programs cannot run at all.

On non-CICS, calls between VS COBOL II NORES programs (that is, programs compiled with the NORES compiler option) and Enterprise COBOL are not supported. On CICS,, VS COBOL II NORES programs cannot run at all.

On non-CICS calls and on CICS, any calls between VS COBOL II RES programs and Enterprise COBOL programs are supported. For additional details, see the *Enterprise COBOL Programming Guide*.

For a complete list of calls between COBOL and assembler (including whether they are supported or not when running with Language Environment), see “Runtime support for assembler COBOL calls under CICS” on page 259.

Can you convert programs selectively to Enterprise COBOL?

Yes, unless an application contains any OS/VS COBOL programs. When you convert applications containing OS/VS COBOL programs, you must convert all of the OS/VS COBOL programs in the run unit to Enterprise COBOL.

We have had errors when running COBOL programs 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 Enterprise COBOL?

No, for QSAM you can turn off automatic file creation with the Language Environment CBLQDA(OFF) runtime option.

When should you use the CMPR2 option?

The CMPR2/NOCMPR2 option is not available in Enterprise COBOL. Enterprise COBOL behaves as if NOCMPR2 were in effect at all times. Any programs that were compiled with CMPR2 with a previous compiler must be upgraded to the COBOL 1985 standard to compile with Enterprise COBOL.

For more details, see “Migrating from the CMPR2 compiler option to NOCMPR2” on page 107.

Is the signature area of Enterprise COBOL programs the same as for OS/VS COBOL and VS COBOL II?

No, but a map of the signature area is in the *Enterprise 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, and so on.

Compiling with Enterprise COBOL

Can you compile programs written for OS/VS COBOL with Enterprise COBOL using the CMPR2 option?

No, CMPR2 is not available with Enterprise COBOL.

For additional details, see “Upgrading your source to Enterprise COBOL” on page 13.

Can you compile programs written for VS COBOL II with Enterprise COBOL?

Yes. For additional details, see “Upgrading your source to Enterprise COBOL” on page 13.

What utilities or tools can assist in converting OS/VS COBOL or VS COBOL II source to Enterprise 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 Enterprise COBOL source:

1. The COBOL conversion aid (CCCA), which is included with the IBM Debug Tool product, assists in converting OS/VS COBOL and VS COBOL II source to Enterprise COBOL source.
2. The COBOL Report Writer Precompiler 5798-DYR assists in converting OS/VS COBOL Report Writer code, or allows you to continue using it.
3. The Debug Tool Load Module Analyzer can determine the language translator for each object in your load modules. The Debug Tool Load Module Analyzer is included with the IBM Debug Tool product.
4. The Edge Portfolio Analyzer can provide assistance in taking an inventory of your existing load modules by reporting the compiler, compiler release, and compiler options used.

The Edge Portfolio Analyzer is no longer sold by IBM. For more information about the Edge Portfolio Analyzer, see www.edge-information.com.

5. WebSphere® Studio Asset Analyzer, product number 5655-M22, assists in taking an inventory, and analyzing the impact that code changes make upon your enterprise assets.

Does Enterprise COBOL meet COBOL 85 Standard?

Yes, Enterprise COBOL supports all required modules of COBOL 85 Standard at the highest level defined by the Standard.

Does Enterprise COBOL meet the COBOL 2002 Standard?

Enterprise COBOL supports many parts of the COBOL 2002 Standard.

Binding (link-editing) Enterprise COBOL programs

What is the difference between an object module, a load module, and a program object?

An object module is the output of the compiler and input to the binder. A load module is a non-GOFF executable that is output from the binder with an Enterprise COBOL V4 or earlier object module. A program object is a new style GOOF executable that is the output from the binder when binding an object module from Enterprise COBOL V5.1.

Are PDS and PDSE data sets allowed for object and load modules with Enterprise COBOL V5?

Compiler output data sets can be PDS or PDSE, including the object module. The output of the bind step must be a PDSE. When COBOL object modules are linked (bound) they become program objects and must be stored in PDSE data sets.

Language Environment services

What is COBOL multithreading and how does it relate to PL/I multitasking?

COBOL multithreading is the support of multiple programs running at the same time in the same address space in the same process. It can be initiated by COBOL programs calling `pthread_create` or C programs doing "pthread create". It is compatible with PL/I multitasking in that multiple PL/I tasks can call COBOL programs when they are compiled with the `THREAD` compiler option.

PL/I can initiate multitasking using native language and manage the interaction between the separate tasks.

Note: COBOL multithreading is not related to the CICS concept of 'threadsafe'.

Language Environment runtime options

Does Enterprise COBOL V5.1.1 use HEAP for WORKING-STORAGE?

It uses HEAP for WORKING-STORAGE when the COBOL program is compiled with the `RENT` option and is in one of the following cases:

- Compiled with the `DATA(24)` compiler option
- Running in CICS
- A COBOL V5.1.1 in a program object that contains only COBOL programs (V5.1.1, V4.2 or earlier) and assembly programs. There are no Language Environment interlanguage calls within the program object and no COBOL V5.1.0 programs.

- A COBOL V5 program in a program object where the main entry point is COBOL V5. In this case, the program object can contain Language Environment interlanguage calls, with COBOL statically linking with C, C++ or PL/I. All COBOL V5 programs within such program objects (even if they are not the main entry point) have their WORKING-STORAGE allocated from heap storage.

Will lower HEAP storage values for COBOL performance affect the performance of C or C++ programs?

Yes. If the C programs use a lot of MALLOC statements, 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 Enterprise COBOL use STACK storage?

Enterprise COBOL programs use STACK storage for LOCAL-STORAGE data items. Other COBOL programs do not use STACK storage.

COBOL runtime routines do use STACK 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, and 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.

Subsystems

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 that the programs must be VS COBOL II or Enterprise COBOL because 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.

Under CICS, mixed VS COBOL II and PL/I is not allowed.

Do I need to specify the TRAP(OFF) runtime option when using the CBLTDLI interface under IMS?

No, TRAP(OFF) is not supported 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*.

z/OS

Does COBOL run in 64-bit z/OS?

Yes. Though COBOL does not yet support 64-bit addressing in COBOL programs, you will get some of the benefits of 64-bit z/OS just by moving to it. With a 64-bit addressable real memory backing your virtual memory, there will be less paging and swapping and therefore better system performance, and you don't have to change your programs at all! In addition, DB2 can exploit 64-bit addressing for SQL statements in COBOL programs without any changes to the COBOL programs.

Even when your z/OS system is running in 64-bit mode, you can still run existing AMODE 24 and AMODE 31 applications without having to relink or recompile them. You can get improved system performance without any changes to your applications.

Performance

Is there a CPU savings when one converts from OS/VS COBOL to Enterprise COBOL?

Yes. Enterprise COBOL V5.1 can give you a significant performance improvement when compared to all older COBOL compilers. It is especially true for programs with a lot of arithmetic.

Service

Do I need to recompile all of my programs to get IBM service support for my applications?

If 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 18.

Object-oriented syntax, and Java 5, Java 6 and Java 7 SDKs

How do I run existing COBOL applications with Java 5, Java 6, or Java 7?

Earlier versions of Enterprise COBOL applications that use object-oriented syntax for Java interoperability are supported with Java SDK 1.4.2.

To run these existing applications with Java 5, Java 6, or Java 7, do these steps:

1. Recompile and relink the applications using Enterprise COBOL Version 4 Release 2 or later.
2. Recompile the generated Java class that is associated with each object-oriented COBOL class using the javac command from Java 5, Java 6, or Java 7.

Appendix B. COBOL reserved word comparison

The following table shows differences between in reserved words OS/VS COBOL, VS COBOL II, IBM COBOL, and Enterprise COBOL.

Information about source language comparison can be found in:

- Chapter 5, “Upgrading OS/VS COBOL source programs,” on page 43
- Chapter 7, “Upgrading VS COBOL II source programs,” on page 93
- Chapter 9, “Upgrading IBM COBOL source programs,” on page 101
- Chapter 11, “Upgrading programs from Enterprise COBOL Version 3,” on page 147
- Chapter 13, “Upgrading from Enterprise COBOL Version 4,” on page 165

Bold text indicates new reserved words (excluding new words reserved for future development) that have been added since IBM COBOL.

Key:

- X** The word is reserved in the product.
- X*** Within the IBM COBOL column, the word is reserved in COBOL for OS/390 & VM, Version 2 Release 2 and later only. It is not reserved in Version 2 Release 1 or earlier.
- X**** Within the Enterprise COBOL column, the word is reserved in Enterprise COBOL Version 4 Release 1 and later only. It is not reserved in Enterprise COBOL Version 3 or earlier.
- X***** Within the Enterprise COBOL column, the word is reserved in Enterprise COBOL Version 4 Release 2 and later. It is not reserved in Enterprise COBOL Version 4 Release 1 or earlier.
- X****** Within the Enterprise COBOL column, the word is reserved in Enterprise COBOL Version 5 Release 1. It is not reserved in Enterprise COBOL Version 4 Release 2 or earlier.
- The word is *not* reserved in the product. (This includes obsolete reserved words that are no longer flagged.)
- CDW** The word is an Enterprise 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.
- UNS** The word is a COBOL 85 Standard reserved word for a feature not supported by this compiler. For some of these words, the feature is supported by the Report Writer Precompiler. If used in a program, it is recognized as a reserved word and flagged with a severe message.

Table 40. Reserved word comparison

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
ACCEPT	X	X	X	X
ACCESS	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
ACTIVE-CLASS	RFD	-	-	-
ACTUAL	-	-	-	X
ADD	X	X	X	X
ADDRESS	X	X	X	X
ADVANCING	X	X	X	X
AFTER	X	X	X	X
ALIGNED	RFD	-	-	-
ALL	X	X	X	X
ALLOCATE	RFD	-	-	-
ALPHABET	X	X	X	-
ALPHABETIC	X	X	X	X
ALPHABETIC-LOWER	X	X	X	-
ALPHABETIC-UPPER	X	X	X	-
ALPHANUMERIC	X	X	X	-
ALPHANUMERIC-EDITED	X	X	X	-
ALSO	X	X	X	X
ALTER	X	X	X	X
ALTERNATE	X	X	X	X
AND	X	X	X	X
ANY	X	X	X	-
ANYCASE	RFD	-	-	-
APPLY	X	X	X	X
ARE	X	X	X	X
AREA	X	X	X	X
AREAS	X	X	X	X
ASCENDING	X	X	X	X
ASSIGN	X	X	X	X
AT	X	X	X	X
AUTHOR	X	X	X	X
AUTOMATIC	RFD	-	-	-
B-AND	RFD	RFD	RFD	-
B-NOT	RFD	RFD	RFD	-
B-OR	RFD	RFD	RFD	-
B-XOR	RFD	-	-	-
BASED	RFD	-	-	-
BASIS	CDW	CDW	CDW	X
BEFORE	X	X	X	X
BEGINNING	X	X	X	X
BINARY	X	X	X	-

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
BINARY-CHAR	RFD	-	-	-
BINARY-DOUBLE	RFD	-	-	-
BINARY-LONG	RFD	-	-	-
BINARY-SHORT	RFD	-	-	-
BIT	RFD	RFD	RFD	-
BLANK	X	X	X	X
BLOCK	X	X	X	X
BOOLEAN	RFD	RFD	RFD	-
BOTTOM	X	X	X	X
BY	X	X	X	X
CALL	X	X	X	X
CANCEL	X	X	X	X
CBL	CDW	CDW	CDW	X
CD	UNS	UNS	UNS	X
CF	UNS	UNS	UNS	X
CH	UNS	UNS	UNS	X
CHANGED	-	-	-	X
CHARACTER	X	X	X	X
CHARACTERS	X	X	X	X
CLASS	X	X	X	-
CLASS-ID	X	X	-	-
CLOCK-UNITS	UNS	UNS	UNS	-
CLOSE	X	X	X	X
COBOL	X	X	X	-
CODE	X	X	X	X
CODE-SET	X	X	X	X
COL	RFD	-	-	-
COLLATING	X	X	X	X
COLS	RFD	-	-	-
COLUMN	UNS	UNS	UNS	X
COLUMNS	RFD	-	-	-
COM-REG	X	X	X	-
COMMA	X	X	X	X
COMMON	X	X	X	-
COMMUNICATION	UNS	UNS	UNS	X
COMP	X	X	X	X
COMP-1	X	X	X	X
COMP-2	X	X	X	X
COMP-3	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
COMP-4	X	X	X	X
COMP-5	X	X*	RFD	-
COMPUTATIONAL	X	X	X	X
COMPUTATIONAL-1	X	X	X	X
COMPUTATIONAL-2	X	X	X	X
COMPUTATIONAL-3	X	X	X	X
COMPUTATIONAL-4	X	X	X	X
COMPUTATIONAL-5	X	X*	RFD	
COMPUTE	X	X	X	X
CONDITION	RFD	-	-	-
CONFIGURATION	X	X	X	X
CONSOLE	-	-	-	X
CONSTANT	RFD	-	-	-
CONTAINS	X	X	X	X
CONTENT	X	X	X	-
CONTINUE	X	X	X	-
CONTROL	UNS	UNS	UNS	X
CONTROLS	UNS	UNS	UNS	X
CONVERTING	X	X	X	-
COPY	CDW	CDW	CDW	X
CORR	X	X	X	X
CORRESPONDING	X	X	X	X
COUNT	X	X	X	X
CRT	RFD	-	-	-
CSP	-	-	-	X
CURRENCY	X	X	X	X
CURRENT-DATE	-	-	-	X
CURSOR	RFD	-	-	-
C01	-	-	-	X
C02	-	-	-	X
C03	-	-	-	X
C04	-	-	-	X
C05	-	-	-	X
C06	-	-	-	X
C07	-	-	-	X
C08	-	-	-	X
C09	-	-	-	X
C10	-	-	-	X
C11	-	-	-	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
C12	-	-	-	X
DATA	X	X	X	X
DATA-POINTER	RFD	-	-	-
DATE	X	X	X	X
DATE-COMPILED	X	X	X	X
DATE-WRITTEN	X	X	X	X
DAY	X	X	X	X
DAY-OF-WEEK	X	X	X	-
DBCS	X	X	X	-
DE	UNS	UNS	UNS	X
DEBUG	-	-	-	X
DEBUG-CONTENTS	X	X	X	X
DEBUG-ITEM	X	X	X	X
DEBUG-LINE	X	X	X	X
DEBUG-NAME	X	X	X	X
DEBUG-SUB-1	X	X	X	X
DEBUG-SUB-2	X	X	X	X
DEBUG-SUB-3	X	X	X	X
DEBUGGING	X	X	X	X
DECIMAL-POINT	X	X	X	X
DECLARATIVES	X	X	X	X
DEFAULT	RFD	RFD	RFD	-
DELETE	X	X	X	X
DELIMITED	X	X	X	X
DELIMITER	X	X	X	X
DEPENDING	X	X	X	X
DESCENDING	X	X	X	X
DESTINATION	UNS	UNS	UNS	X
DETAIL	UNS	UNS	UNS	X
DISABLE	UNS	UNS	UNS	X
DISP	-	-	-	X
DISPLAY	X	X	X	X
DISPLAY-ST	-	-	-	X
DISPLAY-1	X	X	X	-
DIVIDE	X	X	X	X
DIVISION	X	X	X	X
DOWN	X	X	X	X
DUPLICATES	X	X	X	X
DYNAMIC	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
EC	RFD	-	-	-
EGCS	X	X	X	-
EGI	UNS	UNS	UNS	X
EJECT	CDW	CDW	CDW	X
ELSE	X	X	X	X
EMI	UNS	UNS	UNS	X
ENABLE	UNS	UNS	UNS	X
END	X	X	X	X
END-ACCEPT	RFD	-	-	-
END-ADD	X	X	X	-
END-CALL	X	X	X	-
END-COMPUTE	X	X	X	-
END-DELETE	X	X	X	-
END-DISPLAY	RFD	-	-	-
END-DIVIDE	X	X	X	-
END-EVALUATE	X	X	X	-
END-EXEC	X	X*	-	-
END-IF	X	X	X	-
END-INVOKE	X	X	-	-
END-MULTIPLY	X	X	X	-
END-OF-PAGE	X	X	X	X
END-PERFORM	X	X	X	-
END-READ	X	X	X	-
END-RECEIVE	UNS	UNS	UNS	-
END-RETURN	X	X	X	-
END-REWRITE	X	X	X	-
END-SEARCH	X	X	X	-
END-START	X	X	X	-
END-STRING	X	X	X	-
END-SUBTRACT	X	X	X	-
END-UNSTRING	X	X	X	-
END-WRITE	X	X	X	-
END-XML	X	-	-	-
ENDING	X	X	X	X
ENTER	X	X	X	X
ENTRY	X	X	X	X
ENVIRONMENT	X	X	X	X
EO	RFD	-	-	-
EOP	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
EQUAL	X	X	X	X
ERROR	X	X	X	X
ESI	UNS	UNS	UNS	X
EVALUATE	X	X	X	-
EVERY	X	X	X	X
EXAMINE	-	-	-	X
EXCEPTION	X	X	X	X
EXCEPTION-OBJECT	RFD	-	-	-
EXEC	X	X*	-	-
EXECUTE	X	X*	-	-
EXHIBIT	-	-	-	X
EXIT	X	X	X	X
EXTEND	X	X	X	X
EXTERNAL	X	X	X	-
FACTORY	X	X*	-	-
FALSE	X	X	X	-
FD	X	X	X	X
FILE	X	X	X	X
FILE-CONTROL	X	X	X	X
FILE-LIMIT	-	-	-	X
FILE-LIMITS	-	-	-	X
FILLER	X	X	X	X
FINAL	UNS	UNS	UNS	X
FIRST	X	X	X	X
FLOAT-EXTENDED	RFD	-	-	-
FLOAT-LONG	RFD	-	-	-
FLOAT-SHORT	RFD	-	-	-
FOOTING	X	X	X	X
FOR	X	X	X	X
FORMAT	RFD	RFD	RFD	-
FREE	RFD	RFD	RFD	-
FROM	X	X	X	X
FUNCTION	X	X	-	-
FUNCTION-ID	RFD	-	-	-
FUNCTION-POINTER	X	-	-	-
GENERATE	UNS	UNS	UNS	X
GET	RFD	RFD	RFD	-
GIVING	X	X	X	X
GLOBAL	X	X	X	-

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
GO	X	X	X	X
GOBACK	X	X	X	X
GREATER	X	X	X	X
GROUP	UNS	UNS	UNS	X
GROUP-USAGE	X	-	-	-
HEADING	UNS	UNS	UNS	X
HIGH-VALUE	X	X	X	X
HIGH-VALUES	X	X	X	X
I-O	X	X	X	X
I-O-CONTROL	X	X	X	X
ID	X	X	X	X
IDENTIFICATION	X	X	X	X
IF	X	X	X	X
IN	X	X	X	X
INDEX	X	X	X	X
INDEXED	X	X	X	X
INDICATE	UNS	UNS	UNS	X
INHERITS	X	X	-	-
INITIAL	X	X	X	X
INITIALIZE	X	X	X	-
INITIATE	UNS	UNS	UNS	X
INPUT	X	X	X	X
INPUT-OUTPUT	X	X	X	X
INSERT	CDW	CDW	CDW	X
INSPECT	X	X	X	X
INSTALLATION	X	X	X	X
INTERFACE	RFD	-	-	-
INTERFACE-ID	RFD	-	-	-
INTO	X	X	X	X
INVALID	X	X	X	X
INVOKE	X	X	-	-
IS	X	X	X	X
JNIENVPTR	X	-	-	-
JUST	X	X	X	X
JUSTIFIED	X	X	X	X
KANJI	X	X	X	-
KEY	X	X	X	X
LABEL	X	X	X	X
LAST	UNS	UNS	UNS	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
LEADING	X	X	X	X
LEAVE	-	-	-	X
LEFT	X	X	X	X
LENGTH	X	X	X	X
LESS	X	X	X	X
LIMIT	UNS	UNS	UNS	X
LIMITS	UNS	UNS	UNS	X
LINAGE	X	X	X	X
LINAGE-COUNTER	X	X	X	-
LINE	X	X	X	X
LINE-COUNTER	UNS	UNS	UNS	X
LINES	X	X	X	X
LINKAGE	X	X	X	X
LOCAL-STORAGE	X	X	-	-
LOCALE	RFD	-	-	-
LOCK	X	X	X	X
LOW-VALUE	X	X	X	X
LOW-VALUES	X	X	X	X
MEMORY	X	X	X	X
MERGE	X	X	X	X
MESSAGE	UNS	UNS	UNS	X
METAClass	-	X	-	-
METHOD	X	X	-	-
METHOD-ID	X	X	-	-
MINUS	RFD	-	-	-
MODE	X	X	X	X
MODULES	X	X	X	X
MORE-LABELS	X	X	X	X
MOVE	X	X	X	X
MULTIPLE	X	X	X	X
MULTIPLY	X	X	X	X
NAMED	-	-	-	X
NATIONAL	X	-	-	-
NATIONAL-EDITED	X	-	-	-
NATIVE	X	X	X	X
NEGATIVE	X	X	X	X
NESTED	RFD	-	-	-
NEXT	X	X	X	X
NO	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
NOMINAL	-	-	-	X
NOT	X	X	X	X
NOTE	-	-	-	X
NULL	X	X	X	-
NULLS	X	X	X	-
NUMBER	UNS	UNS	UNS	X
NUMERIC	X	X	X	X
NUMERIC-EDITED	X	X	X	-
OBJECT	X	X	-	-
OBJECT-COMPUTER	X	X	X	X
OBJECT-REFERENCE	RFD	-	-	-
OCCURS	X	X	X	X
OF	X	X	X	X
OFF	X	X	X	X
OMITTED	X	X	X	X
ON	X	X	X	X
OPEN	X	X	X	X
OPTIONAL	X	X	X	X
OPTIONS	RFD	-	-	-
OR	X	X	X	X
ORDER	X	X	X	-
ORGANIZATION	X	X	X	X
OTHER	X	X	X	-
OTHERWISE	-	-	-	X
OUTPUT	X	X	X	X
OVERFLOW	X	X	X	X
OVERRIDE	X	X	-	-
PACKED-DECIMAL	X	X	X	-
PADDING	X	X	X	-
PAGE	X	X	X	X
PAGE-COUNTER	UNS	UNS	UNS	X
PASSWORD	X	X	X	X
PERFORM	X	X	X	X
PF	UNS	UNS	UNS	X
PH	UNS	UNS	UNS	X
PIC	X	X	X	X
PICTURE	X	X	X	X
PLUS	UNS	UNS	UNS	X
POINTER	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
POSITION	X	X	X	X
POSITIONING	-	-	-	X
POSITIVE	X	X	X	X
PRESENT	RFD	RFD	RFD	-
PREVIOUS	RFD	RFD	-	-
PRINT-SWITCH	-	-	-	X
PRINTING	UNS	UNS	UNS	-
PROCEDURE	X	X	X	X
PROCEDURE-POINTER	X	X	-	-
PROCEDURES	X	X	X	X
PROCEED	X	X	X	X
PROCESSING	X	X	X	X
PROGRAM	X	X	X	X
PROGRAM-ID	X	X	X	X
PROGRAM-POINTER	RFD	-	-	-
PROPERTY	RFD	-	-	-
PROTOTYPE	RFD	-	-	-
PURGE	UNS	UNS	UNS	-
QUEUE	UNS	UNS	UNS	X
QUOTE	X	X	X	X
QUOTES	X	X	X	X
RAISE	RFD	-	-	-
RAISING	RFD	-	-	-
RANDOM	X	X	X	X
RD	UNS	UNS	UNS	X
READ	X	X	X	X
READY	X	X	X	X
RECEIVE	UNS	UNS	UNS	X
RECORD	X	X	X	X
RECORD-OVERFLOW	-	-	-	X
RECORDING	X	X	X	X
RECORDS	X	X	X	X
RECURSIVE	X	X	-	-
REDEFINES	X	X	X	X
REEL	X	X	X	X
REFERENCE	X	X	X	-
REFERENCES	X	X	X	X
RELATIVE	X	X	X	X
RELEASE	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
RELOAD	X	X	X	X
REMAINDER	X	X	X	X
REMARKS	-	-	-	X
REMOVAL	X	X	X	X
RENAMES	X	X	X	X
REORG-CRITERIA	-	-	-	X
REPLACE	X	X	X	-
REPLACING	X	X	X	X
REPORT	UNS	UNS	UNS	X
REPORTING	UNS	UNS	UNS	X
REPORTS	UNS	UNS	UNS	X
REPOSITORY	X	X	-	-
REREAD	-	-	-	X
RERUN	X	X	X	X
RESERVE	X	X	X	X
RESET	X	X	X	X
RESUME	RFD	-	-	-
RETRY	RFD	-	-	-
RETURN	X	X	X	X
RETURN-CODE	X	X	X	X
RETURNING	X	X	-	-
REVERSED	X	X	X	X
REWIND	X	X	X	X
REWRITE	X	X	X	X
RF	UNS	UNS	UNS	X
RH	UNS	UNS	UNS	X
RIGHT	X	X	X	X
ROUNDED	X	X	X	X
RUN	X	X	X	X
SAME	X	X	X	X
SCREEN	RFD	-	-	-
SD	X	X	X	X
SEARCH	X	X	X	X
SECTION	X	X	X	X
SECURITY	X	X	X	X
SEEK	-	-	-	X
SEGMENT	UNS	UNS	UNS	X
SEGMENT-LIMIT	X	X	X	X
SELECT	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
SELECTIVE	-	-	-	X
SELF	X	X	-	-
SEND	UNS	UNS	UNS	X
SENTENCE	X	X	X	X
SEPARATE	X	X	X	X
SEQUENCE	X	X	X	X
SEQUENTIAL	X	X	X	X
SERVICE	X	X	X	X
SET	X	X	X	X
SHARING	RFD	-	-	-
SHIFT-IN	X	X	X	-
SHIFT-OUT	X	X	X	-
SIGN	X	X	X	X
SIZE	X	X	X	X
SKIP1	CDW	CDW	CDW	X
SKIP2	CDW	CDW	CDW	X
SKIP3	CDW	CDW	CDW	X
SORT	X	X	X	X
SORT-CONTROL	X	X	X	-
SORT-CORE-SIZE	X	X	X	X
SORT-FILE-SIZE	X	X	X	X
SORT-MERGE	X	X	X	X
SORT-MESSAGE	X	X	X	X
SORT-MODE-SIZE	X	X	X	X
SORT-RETURN	X	X	X	X
SOURCE	UNS	UNS	UNS	X
SOURCE-COMPUTER	X	X	X	X
SOURCES	RFD	-	-	-
SPACE	X	X	X	X
SPACES	X	X	X	X
SPECIAL-NAMES	X	X	X	X
SQL	X	X*	-	-
STANDARD	X	X	X	X
STANDARD-1	X	X	X	X
STANDARD-2	X	X	X	-
START	X	X	X	X
STATUS	X	X	X	X
STOP	X	X	X	X
STRING	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
SUB-QUEUE-1	UNS	UNS	UNS	X
SUB-QUEUE-2	UNS	UNS	UNS	X
SUB-QUEUE-3	UNS	UNS	UNS	X
SUB-SCHEMA	RFD	RFD	RFD	-
SUBTRACT	X	X	X	X
SUM	UNS	UNS	UNS	X
SUPER	X	X	-	-
SUPPRESS	X	X	X	X
SYMBOLIC	X	X	X	X
SYNC	X	X	X	X
SYNCHRONIZED	X	X	X	X
SYSIN	-	-	-	X
SYSLIST	-	-	-	X
SYSOUT	-	-	-	X
SYSPUNCH	X	X	X	X
SYSTEM-DEFAULT	RFD	-	-	-
S01	-	-	-	X
S02	-	-	-	X
TABLE	UNS	UNS	UNS	X
TALLY	X	X	X	X
TALLYING	X	X	X	X
TAPE	X	X	X	X
TERMINAL	UNS	UNS	UNS	X
TERMINATE	UNS	UNS	UNS	X
TEST	X	X	X	-
TEXT	UNS	UNS	UNS	X
THAN	X	X	X	X
THEN	X	X	X	X
THROUGH	X	X	X	X
THRU	X	X	X	X
TIME	X	X	X	X
TIME-OF-DAY	-	-	-	X
TIMES	X	X	X	X
TITLE	CDW	CDW	CDW	-
TO	X	X	X	X
TOP	X	X	X	X
TOTALED	-	-	-	X
TOTALING	-	-	-	X
TRACE	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
TRACK-AREA	-	-	-	X
TRACK-LIMIT	-	-	-	X
TRACKS	-	-	-	X
TRAILING	X	X	X	X
TRANSFORM	-	-	-	X
TRUE	X	X	X	-
TYPE	X	X*	-	-
TYPedef	RFD	-	-	-
UNIT	X	X	X	X
UNIVERSAL	RFD	-	-	-
UNLOCK	RFD	-	-	-
UNSTRING	X	X	X	X
UNTIL	X	X	X	X
UP	X	X	X	X
UPDATE	RFD	RFD	RFD	-
UPON	X	X	X	X
UPSI-0	-	-	-	X
UPSI-1	-	-	-	X
UPSI-2	-	-	-	X
UPSI-3	-	-	-	X
UPSI-4	-	-	-	X
UPSI-5	-	-	-	X
UPSI-6	-	-	-	X
UPSI-7	-	-	-	X
USAGE	X	X	X	X
USE	X	X	X	X
USER-DEFAULT	RFD	-	-	-
USING	X	X	X	X
VAL-STATUS	RFD	-	-	-
VALID	RFD	RFD	RFD	-
VALIDATE	RFD	RFD	RFD	-
VALIDATE-STATUS	RFD	-	-	-
VALUE	X	X	X	X
VALUES	X	X	X	X
VARYING	X	X	X	X
WHEN	X	X	X	X
WHEN-COMPILED	X	X	X	X
WITH	X	X	X	X
WORDS	X	X	X	X

Table 40. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
WORKING-STORAGE	X	X	X	X
WRITE	X	X	X	X
WRITE-ONLY	X	X	X	X
XML	X	-	-	-
XML-CODE	X	-	-	-
XML-EVENT	X	-	-	-
XML-INFORMATION	X****	-	-	-
XML-NAMESPACE	X**	-	-	-
XML-NAMESPACE-PREFIX	X**	-	-	-
XML-NNAMESPACE	X**	-	-	-
XML-NNAMESPACE-PREFIX	X**	-	-	-
XML-NTEXT	X	-	-	-
XML-SCHEMA	X***	-	-	-
XML-TEXT	X	-	-	-
ZERO	X	X	X	X
ZEROES	X	X	X	X
ZEROS	X	X	X	X
-	X***	-	-	-
<	X	X	X	X
<>	RFD			
<=	X	X	X	-
+	X	X	X	X
*	X	X	X	X
**	X	X	X	X
-	X	X	X	X
/	X	X	X	X
>	X	X	X	X
>=	X	X	X	-
=	X	X	X	X
*>	X****	-	-	-
::	RFD			

Appendix C. Conversion tools for source programs

A number of conversion tools are available to help you upgrade OS/VS COBOL, VS COBOL II, or IBM COBOL source programs to Enterprise COBOL.

This appendix describes the conversion tools available for your assistance during the conversion. These tools are:

- MIGR compiler option (OS/VS COBOL)
- Other programs that aid conversion

This appendix helps you to determine which, if any, of the tools to use, and understand how to use them and how to analyze their output to assess the extent of the remaining conversion effort.

MIGR compiler option

You can use the OS/VS COBOL MIGR compiler option when you are planning to convert an OS/VS COBOL program to Enterprise COBOL. This option helps you understand the magnitude of the conversion effort.

MIGR can also ease any planned future conversion by helping you avoid using OS/VS COBOL source language not supported by Enterprise COBOL. By compiling your programs using MIGR, you can determine ahead of time which language elements must be converted.

There are incompatibilities in the following areas:

- New reserved words that are introduced because of COBOL functions that have been added (previously valid user words might now be illegal)
- Language function that is supported in a different manner
- Language function that is not supported

You can set the MIGR compiler option either as an installation default, 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 Enterprise COBOL.

Language differences

The following language differences exist between Enterprise COBOL and OS/VS COBOL.

- Changes to ALPHABETIC class
- B symbol in PICTURE clause
- Changes to CALL statement
- Changes to CBL compiler directing statement
- Changes to Combined abbreviated relation condition
- 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)
- Changes to IF . . . OTHERWISE statement
- 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 . . .
- Changes to OCCURS DEPENDING ON clause
- Changes in intermediate results for ON SIZE ERROR option
- 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.
- Changes to PROGRAM COLLATING SEQUENCE clause
- 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
- Changes to RERUN clause
- Changes to RESERVE clause
- Changes to Reserved word list
- SPECIAL-NAMES: alphabet-name IS xxxxx
- Changes in evaluation for subscripts out of range
- 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 switches 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

Via the link below, you can see OS/VS COBOL statements supported with enhanced accuracy in Enterprise COBOL and flagged by a message indicating that more accurate results might be provided in Enterprise 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 LANTLRVL(1) compiler option, are not supported in Enterprise COBOL and are flagged when the MIGR compiler option is specified.

- COPY language—1968
- JUSTIFIED|JUST clause with VALUE
- Changes in scaling for MOVE statement and comparison
- 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 LANTLRVL(2) statements not supported

The following OS/VS COBOL statements, applicable to both the LANTLRVL(1) and LANTLRVL(2) compiler options, are not supported in Enterprise 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 LANTLRVL 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

The Report Writer Precompiler can convert these statements for you. See “COBOL Report Writer Precompiler” on page 255.

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 results 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:

- The Debug Tool Load Module Analyzer can determine the language translator for each object in your load modules.

The Debug Tool Load Module Analyzer is included in Debug Tool.

- COBOL and CICS/VS Command Level Conversion Aid (CCCA)

The COBOL and CICS/VS Command Level Conversion Aid is included in Debug Tool

- CICS application migration aid
- COBOL Report Writer Precompiler

Rational Asset Analyzer

Rational Asset Analyzer provides tools that generate an inventory of enterprise assets and return an index of the relative effort required to make code changes.

COBOL and CICS/VS Command Level Conversion Aid (CCCA)

The COBOL and CICS/VS Command Level Conversion Aid (CCCA), included with the IBM Debug Tool product, converts CICS and non-CICS source code into source code that can be compiled with Enterprise COBOL.

CCCA is updated for reserved word conversions for Enterprise COBOL Version 5 by the PTF for APAR PM86253.

CCCA is designed to automate identifying incompatible source code and converting it to Enterprise COBOL source. Using CCCA should significantly reduce your conversion effort.

CCCA requires that you have an Enterprise COBOL, IBM COBOL, VS COBOL II, or OS/VS COBOL compiler available when converting CICS programs.

The key CCCA facilities:

- Conversion of most syntax differences between OS/VS COBOL or VS COBOL II programs and Enterprise COBOL programs
- Elimination of conflicts between OS/VS COBOL, VS COBOL II, and IBM COBOL user-defined names and Enterprise 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 or conversion of the BLL (Base Locator for Linkage) section mechanism and references

CCCA is designed so that you can tailor it to fit the needs of your shop. CCCA LCPs (Language Conversion Programs), 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 details, see the *COBOL and CICS/VS Command Level Conversion Aid* manual.

When to use CCCA

If you plan to convert your applications from OS/VS COBOL, VS COBOL II or IBM COBOL to Enterprise 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 Enterprise 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 Enterprise 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 Table 41.

Statements dealing with the secondary BLLs are replaced by CONTINUE.

Table 41. 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 ...

COBOL Report Writer Precompiler

You can use the Report Writer Precompiler, product number 5798-DYR, to compile applications that contain Report Writer statements, or to permanently convert Report Writer statements to valid Enterprise COBOL statements.

The Report Writer Precompiler offers the following features:

- Extended Report Writer language capabilities

- Integration with 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 Enterprise 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 Enterprise COBOL compiler

For more detail, see *COBOL Report Writer Precompiler Programmer's Manual* and *COBOL Report Writer Precompiler Installation and Operation*.

Debug Tool Load Module Analyzer

The Debug Tool Load Module Analyzer analyzes MVS™ load modules or program objects to determine the language translator (compiler or assembler) that was used to generate the object for each CSECT.

This program can process all or selected load modules or program objects in a concatenation of PDS or PDSE data sets. Load Module Analyzer is included with the IBM Debug Tool product.

The Edge Portfolio Analyzer

The Edge Portfolio Analyzer helps you inventory your existing load modules by reporting the compiler, compiler release, and compiler options used.

The Edge Portfolio Analyzer is no longer sold by IBM. For more information about the Edge Portfolio Analyzer, see www.edge-information.com.

Appendix D. Applications with COBOL and assembler

If your applications contain mixed COBOL and assembler programs, you might have to make some modifications to the applications.

Do the following tasks as needed:

- Determining requirements for calling and called assembler programs
- Determining which assembler/COBOL calls are supported under non-CICS
- Determining which assembler/COBOL calls are supported under CICS
- Converting programs that change the program mask
- Upgrading applications that use an assembler driver
- Modifying applications in which assembler loads, calls, or deletes COBOL programs

Some information about applications that contain both assembler and COBOL programs is included in other sections of this documentation. For example, you can find information about assembler programs that pass procedure names in “Language elements that changed from OS/VS COBOL” on page 72

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
- Returns to the COBOL caller in the same AMODE that was in use when it was called

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.

Runtime support for assembler COBOL calls under non-CICS

The combinations of calls involving COBOL programs and assembler programs and whether the calls are supported when running under Language Environment under non-CICS are listed in the following table.

For the calls that are *not* supported, Table 42 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.

The term, *IBM COBOL* refers to COBOL/370, COBOL for MVS & VM and COBOL for OS/390 & VM.

Table 42. Language Environment supported calls between COBOL programs and assembler programs under non-CICS; Yes indicates that a call is supported.

Calls from		Issued to						
Call type	Program issuing	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL	LanEnv ¹ Asm ² main	LanEnv ¹ Asm subrtn	Non-LanEnv Asm
Static	Enterprise COBOL	Yes	Yes	Yes	No	No ³	Yes	Yes
	IBM COBOL	Yes	Yes	Yes	Yes	No ³	Yes	Yes
	VS COBOL II with RES	Yes	Yes	Yes	Yes	No ³	Yes ⁴	Yes
	VS COBOL II with NORES	No	Yes	Yes	Yes	No ³	Yes ⁴	Yes
	OS/VS COBOL	No	Yes	Yes	Yes	No ³	Yes ⁴	Yes
Dynamic	Enterprise COBOL	Yes	Yes	Yes	No	No ³	Yes	Yes
	IBM COBOL	Yes	Yes	Yes	Yes	No ³	Yes	Yes
	VS COBOL II with RES	Yes	Yes	Yes	Yes	No ³	Yes	Yes
	VS COBOL II with NORES	No	Yes	Yes	Yes	No ³	Yes	Yes
	OS/VS COBOL	No	Yes	Yes	Yes	No ³	Yes	Yes
VCON	Asm (LanEnv)	Yes	Yes	Yes	Yes	No ³	Yes	Yes
	Asm (non-LanEnv)	Yes	Yes	Yes	Yes	Yes ⁵	No ⁶	Yes
LOAD	Asm (LanEnv)	Yes	Yes	Yes	Yes	No ³	Yes	Yes
BALR	Asm (non-LanEnv)	Yes	Yes	Yes	Yes	Yes ⁵	No ⁶	Yes
LINK	Asm (LanEnv)	Yes	Yes	Yes	Yes ⁷	Yes	No ⁶	Yes
	Asm (non-LanEnv)	Yes	Yes	Yes	Yes ⁷	Yes	No ⁶	Yes

Table 42. Language Environment supported calls between COBOL programs and assembler programs under non-CICS; Yes indicates that a call is supported. (continued)

Calls from		Issued to						
Call type	Program issuing	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL	LanEnv ¹ Asm ² main	LanEnv ¹ Asm subrtn	Non-LanEnv Asm
<p>The failure symptoms described in these notes are as they would occur when the Language Environment TRAP(ON) and ABTERMENC(ABEND) runtime options are in effect.</p> <ol style="list-style-type: none"> 1. (LanEnv stands for Language Environment.) 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. (Asm stands for assembler.) 3. 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. 4. You must specify NAB=NO and MAIN=NO on the CEEENTRY macro. Otherwise, you will receive failure symptom 0C1, 0C4, or 0C5 abend. 5. If the non-Language Environment assembler caller is running within an established Language Environment enclave, see note 3. 6. Failure symptom of 0C1, 0C4, or 0C5 abend. 7. Except when OS/VS COBOL programs exist in another established Language Environment enclave. For detail, see Failure symptom of: message IGZ0005S. 								

Runtime support for assembler COBOL calls under CICS

The combinations of calls involving COBOL programs and assembler programs and whether the calls are supported when running under Language Environment under CICS are listed in the following table.

For the calls that are *not* supported, Table 43 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.

The term *IBM COBOL* refers to COBOL/370, COBOL for MVS & VM, and COBOL for OS/390 & VM.

Table 43. Language Environment supported calls between COBOL programs and assembler programs that run under CICS; Yes indicates that a call is supported.

Calls from		Issued to					
Call type	Program issuing	Enterprise COBOL	IBM COBOL	VS COBOL II	LanEnv ¹ Asm ² main	LanEnv ¹ Asm subtrtn	Non-LanEnv Asm
Static	Enterprise COBOL	Yes	Yes	Yes	No ³	Yes	Yes
	IBM COBOL	Yes	Yes	Yes	No ³	No ⁴	Yes
	VS COBOL II	Yes	Yes	Yes	No ³	No ⁴	Yes
Dynamic	Enterprise COBOL	Yes	Yes	Yes	No ³	Yes	Yes
	IBM COBOL	Yes	Yes	Yes	No ³	Yes	Yes
	VS COBOL II	Yes	Yes	Yes	No ³	Yes	Yes

Table 43. Language Environment supported calls between COBOL programs and assembler programs that run under CICS; Yes indicates that a call is supported. (continued)

Calls from		Issued to					
Call type	Program issuing	Enterprise COBOL	IBM COBOL	VS COBOL II	LanEnv ¹ Asm ² main	LanEnv ¹ Asm subrtn	Non-LanEnv Asm
EXEC CICS LINK	Enterprise COBOL	Yes	Yes	Yes	No ³	No ⁴	Yes
	IBM COBOL	Yes	Yes	Yes	No ³	No ⁴	Yes
	VS COBOL II	Yes	Yes	Yes	No ³	No ⁴	Yes
VCON	Asm (LanEnv)	Yes	Yes	No ⁴	No ³	Yes	Yes
	Asm (non-LanEnv)	No ⁴	No ⁴	No ⁴	No ³	No ⁴	Yes
EXEC CICS LINK	Asm (non-LanEnv)	Yes	Yes	Yes	No ³	No ⁴	Yes
	Asm (non-LanEnv)	Yes	Yes	Yes	No ³	No ⁴	Yes
<p>The failure symptoms described in these notes are as they would occur when the Language Environment TRAP(ON) and ABTERMENC(ABEND) runtime options are in effect.</p> <ol style="list-style-type: none"> 1. (LanEnv stands for Language Environment.) 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. (Asm stands for assembler.) 3. There is no support for Language Environment-conforming assembler main programs under CICS at a level earlier than CICS TS Version 3. Failure symptom: Unpredictable. The applications might appear to run successfully. 4. Failure symptom of: ASRA abend (caused by type 1 or 5 program check). 							

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 Enterprise 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.

Upgrading applications that use an assembler driver

There are three methods for upgrading applications that use an assembler driver to call COBOL subroutines:

- Convert the assembler driver to a Language Environment-conforming assembler driver.
- Modify the assembler driver to set up the Language Environment environment.
- Use the RTEREUS runtime option if the assembler 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 the 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 about how to make your existing assembler programs Language Environment-conforming, see the *Language Environment Programming Guide*.

Modify the assembler driver

If the assembler driver uses either IGZERRE or ILBOSTP0, it must be modified.

Replace the OS/VS COBOL ILBOSTP0 or IGZERRE routine with the Language Environment CEEPIPI INIT_SUB, CEEPIPI INIT_MAIN, and CEEPIPI TERM functions. These Language Environment routines have a convenient complementary termination function that was not available with OS/VS COBOL.

Use an unmodified assembler 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 runtime option. (RTEREUS initializes the runtime 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.

Assembler programs that load and BALR to MAIN COBOL programs

You can LOAD and BALR, then BALR again to OS/VS COBOL main programs from assembler. But it is not supported to LOAD and BALR then BALR again to a main program that was compiled by Enterprise COBOL (or any newer compiler) with the NORENT option. If you recompile an OS/VS COBOL program with Enterprise COBOL and the NORENT compiler option, the program will abend with message IGZ0044S. There are several possible solutions:

- Compile with RENT.
- Change the assembler code to DELETE and LOAD again before a subsequent BALR to NORENT COBOL.
- Change the assembler program to be Language Environment-conforming.

Assembler programs that load and delete COBOL programs

Under Language Environment, assembler programs can SVC load and SVC delete load modules that contain any of the following programs:

- OS/VS COBOL programs
- VS COBOL II programs compiled with the NORENT option
- IBM COBOL programs compiled with the NORENT option
- Enterprise COBOL programs compiled with the NORENT option

Restriction: Debug Tool does not support COBOL programs that are in load modules that are deleted by assembler using SVC delete.

Under Language Environment, assembler programs can SVC load but *cannot* SVC delete load modules that contain any of the following programs:

- VS COBOL II programs compiled with the RENT option
- IBM COBOL programs compiled with the RENT option
- Enterprise 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, perform one of the following tasks:

- Have the assembler program statically call a COBOL program that performs the dynamic call and performs the CANCEL.
- Under Language Environment, use the CEEFETCH and CEERELES macros.

Appendix E. Option comparison

The following table describes the Enterprise COBOL V5 compiler options and installation options, and explains how the options compare with those in OS/VS COBOL, VS COBOL II, IBM COBOL, and Enterprise COBOL V3 and V4.

For complete descriptions of the Enterprise COBOL V5 options, see the *Enterprise COBOL Programming Guide* and the *Enterprise COBOL Customization Guide*.

Table 44. Option comparison

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
ADATA			X	X	X	Produces associated data file at compilation. NOADATA is the default. The Enterprise COBOL ADATA option replaces the COBOL/370 EVENTS option.
ADV	X	X	X	X	X	Adds print control byte at beginning of records. ADV is the default.
AFP					X	Controls the compiler usage of the Additional Floating Point (AFP) registers that are provided by z/Architecture processors. AFP(VOLATILE) is the default.
ANALYZE			X**			Causes the compiler to check the syntax of embedded SQL and CICS statements in addition to native COBOL statements.
ALOWCBL		X	X	X	X	Allows PROCESS or CBL statements in source programs. You can only specify this option at installation time. ALOWCBL is the default.
APOST	X	X	X	X	X	Specifies apostrophe (') as delimiter for literals. QUOTE is the default. In Enterprise 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.
ARCH					X	Specifies the machine architecture for which the executable program instructions are to be generated. ARCH(6) is the default.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
I ARITH			X	X	X	<p>Sets the maximum number of digits that you can specify for decimal data and affects the precision of intermediate results. ARITH(COMPAT) is the default.</p> <p>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.</p> <p>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.</p>
AWO		X	X	X	X	Activates APPLY WRITE-ONLY processing for physical sequential files with VB format. NOAWO is the default.
I BLOCK0				X	X	Activates BLOCK CONTAINS 0 clause for all physical sequential files in the program that specify neither BLOCK CONTAINS nor RECORDING MODE U in the file description.
BUF	X					Allocates buffer storage for compiler work data sets. In Enterprise COBOL, the BUFSIZE option replaces the OS/VS COBOL BUF option.
BUFSIZE		X	X	X	X	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.
I CICS			X	X	X	Enables the integrated CICS translator capability and specifies CICS options. NOCICS is the default.
CLIST	X					<p>Produces a condensed PROCEDURE DIVISION listing plus tables and program statistics. NOCLIST is the default.</p> <p>The VS COBOL II, IBM COBOL, and Enterprise COBOL OFFSET option replaces the OS/VS COBOL CLIST option.</p>

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
CMPR2		X	X			<p>Specified generation of IBM COBOL source code compatible with VS COBOL II Release 2 or other VS COBOL II CMPR2 behavior.</p> <p>NOCMPR2 is the default behavior which cannot be changed. NOCMPR2 specifies the full use of all IBM COBOL language features (including language extensions for object-oriented COBOL and improved interoperability with C programs).</p>
CODEPAGE				X	X	<p>Specifies the code page used for encoding contents of alphanumeric and DBCS data items at run time as well as alphanumeric, national, and DBCS literals in a COBOL source program. CODEPAGE(1140) is the default.</p>
COMPILE		X	X	X	X	<p>Requests an unconditional full compilation. Other options are NOCOMPILE and NOCOMPILE(W E S). The default is NOCOMPILE(S).</p> <p>NOCOMPILE specifies unconditional syntax checking. NOCOMPILE(W E S) specify conditional syntax checking based on the severity of the error.</p> <p>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.</p>
COUNT	X					<p>Produces verb execution summaries at the end of program execution. Each verb is identified by procedure-name and by statement number, and the number of times it was used is indicated.</p> <p>A similar function is provided with Debug Tool.</p>

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
CURRENCY			X	X	X	<p>Defines the 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.</p> <p>NOCURRENCY is the default and indicates that no alternate default currency sign is provided by the CURRENCY option.</p>
DATA(24) DATA(31)		X	X	X	X	<p>Specifies whether reentrant program data areas are acquired above or below the 16-MB line. With DATA(24), reentrant programs data is acquired below the 16-MB line. With DATA(31), reentrant programs data is acquired above the 16-MB line. DATA(31) is the default.</p>
DATEPROC			X	X		<p>Enables the millennium language extensions of the COBOL compiler. Options consist of DATEPROC(FLAG), DATEPROC(NOFLAG), DATEPROC(TRIG), DATEPROC(NOTRIG) and NODATEPROC.</p>
DBCS		X	X	X	X	<p>Tells the compiler to recognize DBCS shift-in and shift-out codes.</p> <p>DBCS is the default.</p>
DBCSXREF=code		X	X	X	X	<p>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.</p> <p>DBCSXREF=NO is the default.</p>
DECK	X	X	X	X	X	<p>Generates object code as 80-character card images and places it in SYSPUNCH file. NODECK is the default.</p>

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
DIAGTRUNC			X	X	X	Causes the compiler to issue a severity-4 (warning) diagnostic message for MOVE statements with numeric receivers when the receiving data has fewer integer positions than the sending data item or literal. NODIAGTRUNC is the default.
DISPSIGN					X	Controls output formatting for DISPLAY of signed numeric items. DISPSIGN(COMPAT) is the default.
DLL			X	X	X	Enables the compiler to generate an object module that is enabled for DLL (Dynamic Link Library) support. NODLL is the default.
DMAP	X					Produces a listing of the DATA DIVISION and implicitly declared items. NODMAP is the default. The VS COBOL II, IBM COBOL, and Enterprise COBOL MAP option replaces the OS/VS COBOL DMAP option.
DUMP	X	X	X	X	X	Specifies that a system dump be produced at end of compilation. NODUMP is the default.
DYNAM	X	X	X	X	X	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) EXIT(MSG-id)		X	X	X	X	Allows the compiler to accept user-supplied modules. (Each <i>string</i> is an optional user-supplied input string to the exit module, and each <i>mod</i> names a user-supplied exit module.) The ADT-id suboption is only available with COBOL for MVS & VM and later compilers. NOEXIT is the default.
EXPORTALL			X	X	X	Instructs the compiler to automatically export certain symbols when the object deck is link-edited to form a DLL. NOEXPORTALL is the default.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
FASTSRT		X	X	X	X	Specifies fast sorting by the IBM DFSORT licensed program. NOFASTSRT is the default, and specifies that Enterprise COBOL will do SORT or MERGE input/output.
FLAG	X	X	X	X	X	Specifies that syntax messages are produced at the level indicated. For OS/VS COBOL the FLAG options are: FLAGW and FLAGE. For Enterprise 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. For Enterprise COBOL, FLAG(I,I) is the default.
FLAGMIG		X	X	X		Specifies NOCMR2 flagging for possible semantic changes from VS COBOL II Release 2 or other programs with CMR2 behavior.
FLAGMIG4				X***		APAR PM93450 for Enterprise COBOL Version 4 Release 2 adds option FLAGMIG4 to identify language elements in Enterprise COBOL Version 4 programs that are not supported, or that are supported differently in Enterprise COBOL Version 5. The compiler will generate a warning diagnostic messages for all such language elements.
FLAGSTD		X	X	X	X	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 C interoperability, and use of the PGMNAME(LONGMIXED) compiler option. NOFLAGSTD is the default.
FDUMP		X				Produces a dump with debugging information when an application ends with an abend. NOFDUMP is the default. The Enterprise COBOL TEST option replaces the VS COBOL II FDUMP option.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
HGPR					X	Controls the compiler usage of the 64-bit registers provided by z/Architecture processors. HGPR(PRESERVE) is the default.
IDLGEN			X			In addition to the normal compile of the COBOL source file, IDLGEN generates IDL definitions for defined classes. NOIDLGEN is the default.
INTDATE			X	X	X	Determines the starting date for integer format dates when used with date intrinsic functions. INTDATE(ANSI) uses COBOL 85 Standard starting date, where Day 1 = January 1, 1601. INTDATE(LILIAN) uses the Language Environment Lilian starting date, where Day 1 = October 15, 1582. INTDATE(ANSI) is the default.
LANGUAGE		X	X	X	X	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	X	X	X	X		Specifies that the program uses the COPY library.
LINECNT=nn	X					Specifies the number of lines per page on the output listing. For VS COBOL II, IBM COBOL, and Enterprise COBOL, the LINECOUNT compiler option replaces the OS/VS COBOL LINECNT option.
LINECOUNT		X	X	X	X	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.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
LIST		X	X	X	X	Produces a listing of assembler language expansion of source code. NOLIST is the default. LIST replaces the OS/VS COBOL PMAP option.
LOAD	X					Stores object code on disk or tape for input to linkage editor. NOLOAD is default. The VS COBOL II, IBM COBOL, and Enterprise COBOL OBJECT option replaces the OS/VS COBOL LOAD option.
MAP		X	X	X	X	Produces a listing of the DATA DIVISION and, implicitly, declared items. NOMAP is the default. MAP replaces the OS/VS COBOL DMAP option.
MAXPCF(<i>n</i>)					X	Instructs the compiler not to optimize code if the program contains a complexity factor greater than <i>n</i> . The default is MAXPCF(60000).
MDECK				X	X	Causes output from the library processing (the expansion of COPY, BASIS, REPLACE, and EXEC SQL INCLUDE statements) to be written to a file. NOMDECK is the default.
NAME	X	X	X	X	X	Indicates that a linkage-editor NAME statement is appended to each object module created. For VS COBOL II, IBM COBOL, and Enterprise COBOL, NAME has the suboptions (ALIAS NOALIAS). If ALIAS is specified, an ALIAS statement is also generated for each ENTRY statement NONAME is the default.
NSYMBOL				X	X	Controls the interpretation of the "N" symbol used in literals and picture clauses, indicating whether national or DBCS processing is assumed. NSYMBOL(NATIONAL) is the default.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
NUM	X					Prints line numbers in error messages and listings. NONUM is the default. The VS COBOL II, IBM COBOL, and Enterprise COBOL NUMBER option replaces the OS/VS COBOL NUM option.
NUMBER		X	X	X	X	Prints line numbers in error messages and listings. NONUMBER is the default. The NUMBER option replaces the OS/VS COBOL NUM option.
NUMCLS		X	X	X	X	Determines, together with the NUMPROC option, valid sign configurations for numeric items in the NUMERIC class test. NUMCLS has two suboptions: (PRIM/ALT). NUMCLS(PRIM) is the default. You can specify NUMCLS only at installation time. For more information, see the: <ul style="list-style-type: none"> Enterprise COBOL Customization Guide
NUMPROC		X	X	X	X	Handles packed/zoned decimal signs as follows: NUMPROC(PFD) Decimal fields assumed to have standard S/390® signs NUMPROC(NOPFD) The compiler does any necessary sign conversion of nonpreferred but valid signs. NUMPROC(MIG) Enterprise COBOL processes sign conversion in a manner very similar to OS/VS COBOL. This suboption is not supported in Enterprise COBOL V5. NUMPROC(NOPFD) is the default.
OBJECT		X	X	X	X	Stores object code on disk or tape for input to linkage editor. NOOBJECT is the default. OBJECT replaces the OS/VS COBOL LOAD option.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
OFFSET		X	X	X	X	Produces a condensed PROCEDURE DIVISION listing plus tables and program statistics. NOOFFSET is the default. OFFSET replaces the OS/VS COBOL CLIST option.
OPTFILE				X	X	Specifies that compiler options should be read from a separate data set or file specified by a SYSOPTF DD statement. OPTFILE is not in effect by default.
OPTIMIZE	X	X	X	X	X	Optimizes the object program. With IBM COBOL and Enterprise COBOL prior to V5, OPTIMIZE had the suboptions of (STD/FULL). The default was NOOPTIMIZE. In Enterprise COBOL V5, OPTIMIZE has the suboptions of (0 / 1 / 2). The OPTIMIZE option specifies increasing levels of optimization to improve application runtime performance. OPTIMIZE(0) is the default.
OUTDD(SYSOUT) OUTDD(ddname)		X	X	X	X	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			X	X	X	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 handled in a manner compatible with older versions of COBOL compilers. PGMNAME(COMPAT) is the default.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
PMAP	X					Produces a listing of assembler language expansion of source code. The VS COBOL II, IBM COBOL, and Enterprise COBOL LIST compiler option replaces the OS/VS COBOL PMAP option.
QUOTE	X	X	X	X	X	Specifies a quotation mark (") as the delimiter for literals. QUOTE is the default. In Enterprise 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	X	X				Causes most library routines to be loaded dynamically, instead of being link-edited with the COBOL program. RES is the default behavior and is not changeable.
RENT		X	X	X	X	Specifies reentrant code in object program. RENT is the default.
RMODE(AUTO) RMODE(24) RMODE(ANY)			X	X	X	Establishes the residency mode for the generated object program. Programs compiled with NORENT will have RMODE(24). Programs compiled with RENT will have RMODE(ANY). RMODE(AUTO) is the default.
SEQ	X					Checks ascending sequencing of source statement line numbers. The VS COBOL II, IBM COBOL, and Enterprise COBOL SEQUENCE option replaces the OS/VS COBOL SEQ option.
SEQUENCE		X	X	X	X	Checks ascending sequencing of source statement line numbers. SEQUENCE is the default. SEQUENCE replaces the OS/VS COBOL SEQ option.
SIZE(MAX) SIZE(nnnnn) SIZE(nnnK)		X	X	X	X, SIZE(MAX) not supported	Specifies virtual storage to be used for compilation. SIZE(5000k) is the default.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
SOURCE	X	X	X	X	X	Produces a listing of the source program and embedded messages. SOURCE is the default.
SPACE	X	X	X	X	X	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 Enterprise COBOL is: SPACE(1), SPACE(2), SPACE(3). SPACE(1) is the default.
SQL			X	X	X	Enables the DB2 coprocessor capability and specifies DB2 suboptions. NOSQL is the default.
SQLIMS					X	Enables the IMS SQL coprocessor capability and specifies IMS suboptions. NOSQLIMS is the default.
SQLSSCID				X	X	Determines whether the CODEPAGE compiler option influences the processing of SQL statements in COBOL programs. Has an effect only when the integrated DB2 coprocessor (SQL compiler option) is used. NOSQLCCSID is the default.
SSRANGE		X	X	X	X	At run time, checks validity of subscript, index, and reference modification references NOSSRANGE is the default.
STGOPT					X	Controls storage optimization. NOSTGOPT is the default.
SYSx	X					Routes DISPLAY output to SYSOUT or to a specified data set. The VS COBOL II, IBM COBOL, and Enterprise COBOL OUTDD option replaces the OS/VS COBOL SYSx option.
STATE	X					Produces a dump with debugging information when an application ends with an abend. The IBM Enterprise COBOL TEST option replaces the OS/VS COBOL STATE option.

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
SUPMAP SYNTAX CSYNTAX	X					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, IBM COBOL, and Enterprise COBOL COMPILE option replaces the OS/VS COBOL SYNTAX, CSYNTAX, and CSUPMAP options.
SYMDMP	X					Produces a symbolic dump. ABEND dumps and dynamic dumps are available through Language Environment services. Symbolic dumps are available by using the TEST compiler option.
SXREF	X					Produces sorted cross-reference listing of data names and procedure names used in program. The VS COBOL II, IBM COBOL, and Enterprise COBOL XREF option replaces the OS/VS COBOL SXREF option.
TERM	X					Sends progress messages to the SYSTERM data set. The VS COBOL II, IBM COBOL, and Enterprise COBOL TERMINAL option replaces the OS/VS COBOL TERM option.
TERMINAL		X	X	X	X	Sends progress messages to the SYSTERM data set. NOTERMINAL is the default. TERMINAL replaces the OS/VS COBOL TERM option.
TEST	X	X	X	X	X	Produces object code usable by Debug Tool for the product. NOTEST(NODWARF) is the default. For details about the suboptions for the Enterprise COBOL TEST option, see the <i>Enterprise COBOL Programming Guide</i> .

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
THREAD				X	X	Enables a COBOL program for execution in a run unit with multiple POSIX threads or PL/I tasks. NOTHREAD is the default.
TRUNC	X	X	X	X	X	<p>Truncates final intermediate results. OS/VS COBOL has the TRUNC and NOTRUNC options (NOTRUNC is the default). VS COBOL II, IBM COBOL, and Enterprise COBOL have the TRUNC(STD OPT BIN) option.</p> <p>TRUNC(STD) Truncates numeric fields according to PICTURE specification of the binary receiving field</p> <p>TRUNC(OPT) Truncates numeric fields in the most optimal way</p> <p>TRUNC(BIN) Truncates binary fields based on the storage they occupy</p> <p>TRUNC(STD) is the default.</p> <p>For a complete description, see the <i>Enterprise COBOL Programming Guide</i>.</p>
TYPECHK			X			<p>Enforces the rules for OO type conformance and issues diagnostics for any violations.</p> <p>NOTYPECHK is the default.</p>
VBREF VBSUM	X	X	X	X	X	<p>Produces a cross-reference listing of all verb types used in program. Only OS/VS COBOL supports VBSUM.</p> <p>NOVBREF is the default.</p>
WORD		X	X	X	X	<p>Tells the 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.</p> <p>NOWORD is the default.</p>

Table 44. Option comparison (continued)

Option	Available in					Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL V3 and V4	Enterprise COBOL V5	
XMLPARSE				X***		For Enterprise COBOL Version 4 only. Selects which XML parser is to be used, either the z/OS XML System Services parser (XMLSS) or the COBOL high-speed parser that was used in Enterprise COBOL Version 3. The default is XMLPARSE(XMLSS).
XREF		X	X	X	X	Produces a sorted cross-reference listing of data names and procedure names used in program. The default is XREF. XREF replaces the OS/VS COBOL SXREF option.
YEARWINDOW			X	X		Specifies the first year of the 100-year window (the century window) to be applied to windowed date field processing by the COBOL compiler. YEARWINDOW(1900) is the default.
ZWB	X	X	X	X	X	Removes the sign from a signed numeric DISPLAY field when comparing it with an alphanumeric field. ZWB is the default.

- X* Available only in COBOL for OS/390 & VM, Version 2 Release 2
- X** Available only in COBOL for OS/390 & VM, Version 2 Release 1 and 2
- X*** Available only in Enterprise COBOL Version 4 Release 1 and 2
- X**** Available only in Enterprise COBOL Version 4 Release 2

Appendix F. Compiler limit comparison

The following table lists the compiler limits for Enterprise COBOL V5, other Enterprise COBOL versions, IBM COBOL, VS COBOL II, and OS/VS COBOL programs.

These are guidelines to the limits in the table:

- Interpret a limit stated in megabytes (MB) as: x megabytes minus 1-B.
- Interpret a limit stated in kilobytes (KB) as: x kilobytes minus 1-B.
- Interpret a limit stated in gigabytes (GB) as: x gigabytes minus 1-B.
- B stands for bytes.
- N/L stands for no limit.
- Footnotes are at the end of the table.

Language element	Enterprise COBOL V5	Other Enterprise COBOL versions	IBM COBOL and VS COBOL II	OS/VS COBOL
Size of program	999,999 lines	999,999 lines	999,999 lines	999,999 lines
Number of literals	4,194,303-B ¹	4,194,303-B ¹	4,194,303-B ¹	16,384-B
Total length of literals	4,194,303-B ¹	4,194,303-B ¹	4,194,303-B ¹	32,767-B after OPT
Reserved word table entries	1536	1536	1536	N/L
COPY REPLACING . . . BY . . . (items per COPY statement)	N/L	N/L	N/L	150
Number of COPY libraries	N/L	N/L	N/L	N/L
Block size of COPY library	32,760-B	32,767-B	32,767-B	16,384-B
IDENTIFICATION DIVISION				
ENVIRONMENT DIVISION				
CONFIGURATION SECTION				
SPECIAL-NAMES paragraph				
<i>mnemonic-name</i> IS	18	18	18	18
UPSI- <i>n</i> . . . (switches)	0-7	0-7	0-7	0-7
<i>alphabet-name</i> IS . . .	N/L	N/L	N/L	N/L
literal THRU . . . or ALSO . . .	256	256	256	256
INPUT-OUTPUT SECTION				
FILE-CONTROL paragraph				
SELECT <i>file-name</i> . . .	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 65,535 file names can be assigned external names	A maximum of 65,535 file names can be assigned external names
ASSIGN <i>system-name</i> . . .	N/L	N/L	N/L	N/L
ALTERNATE RECORD KEY <i>data-name</i> . . .	253	253	253	253

Language element	Enterprise COBOL V5	Other Enterprise COBOL versions	IBM COBOL and VS COBOL II	OS/VS COBOL
RECORD KEY length	N/L ²	N/L ²	N/L ²	255
RESERVE <i>integer</i> (buffers)	255 ³	255 ³	255 ³	255 ³
I-O-CONTROL paragraph				
RERUN ON <i>system-name</i> . . .	32,767	32,767	32,767	32,767
RERUN <i>integer</i> RECORDS	16,777,215	16,777,215	16,777,215	16,777,215
SAME RECORD AREA	255	255	255	255
SAME RECORD AREA FOR <i>file-name</i> . . .	255	255	255	255
SAME SORT/MERGE AREA	N/L ⁴	N/L ⁴	N/L ⁴	N/L ⁴
MULTIPLE FILE <i>file-name</i> . . .	N/L ⁴	N/L ⁴	N/L ⁴	N/L ⁴
DATA DIVISION				
77 data item size	999,999,999	134,217,727	16,777,215	1-MB
Total 01 + 77 (data items)	N/L	N/L	N/L	255
88 condition-names . . .	N/L	N/L	N/L	N/L
66 RENAMES . . .	N/L	N/L	N/L	N/L
PICTURE clause, number of characters in <i>character-string</i>	50	50	30	30
PICTURE clause, numeric item digit positions	With ARITH(COMPAT): 18 With ARITH(EXTEND): 31	18 (or 31) ⁶	For IBM COBOL: 18 (or 31) ⁶ For VS COBOL II: 18	18
PICTURE clause, numeric-edited character positions	249	249	249	127
PICTURE symbol replication ()	999,999,999	134,217,727	16,777,215	99,999
PICTURE symbol replication (), class DBCS items	499,999,999	67,108,863	8,388,607	N/A
PICTURE symbol replication (), class national items	499,999,999	67,108,863	N/A	N/A
PICTURE symbol replication (editing)	32,767	32,767	32,767	99,999
Elementary item size	134,217,727	134,217,727	16,777,215	32,767
OCCURS integer	999,999,999	134,217,727	4,194,303	65,535
Table size	999,999,999	134,217,727	8,388,607	32,767
ASC or DES KEY . . . (per OCCURS clause)	12	12	12	12
Total length of keys (per OCCURS clause)	256B	256B	256B	256B
INDEXED BY . . . (index names by OCCURS clause)	12	12	12	12
Total number of indexes (index names) per class or program	65,535	65,535	65,535	65,535

Language element	Enterprise COBOL V5	Other Enterprise COBOL versions	IBM COBOL and VS COBOL II	OS/VS COBOL
Size of relative index	32,765	32,765	32,765	32,765
FILE SECTION				
FD record description entry	1,048,575	1,048,575	1,048,575	1,048,575
FD <i>file-name</i> . . .	65,535	65,535	65,535	65,535
LABEL <i>data-name</i> . . . (if no optional clauses)	255	255	255	185
Label record length	80-B	80-B	80-B	80-B
DATA RECORD <i>data-name</i> . . .	N/L ⁴	N/L ⁴	N/L ⁴	N/L ⁴
BLOCK CONTAINS <i>integer</i>	2,147,483,647 ⁹	2,147,483,647 ⁹	For IBM COBOL: 2,147,483,647 For VS COBOL II: 1,048,575 ⁵	32,760
RECORD CONTAINS <i>integer</i>	1,048,575 ⁵	1,048,575 ⁵	1,048,575 ⁵	32760
SD <i>file-name</i> . . .	65,535	65,535	65,535	65,535
DATA RECORD <i>data-name</i> . . .	N/L ⁴	N/L ⁴	N/L ⁴	N/L ⁴
WORKING-STORAGE SECTION				
Total size of items without the EXTERNAL attribute	2,147,483,646-B	134,217,727-B	134,217,727-B	1-MB
Total size of items with the EXTERNAL attribute	2,147,483,646-B	134,217,727-B	134,217,727-B	1-MB
LINKAGE SECTION				
Total size	N/L	134,213,631-B	134,217,727-B	1-MB
PROCEDURE DIVISION				
Procedure and constant area	4,194,303 ¹	4,194,303 ¹	4,194,303 ¹	1M+32-KB
PROCEDURE DIVISION USING <i>identifier</i> . . .	32,767	32,767	32,767	N/L
Procedure-names	1,048,575 ¹	1,048,575 ¹	1,048,575 ¹	64-KB ¹
Verbs per line (FDUMP/TEST)	7	7	7	7
Subscripted data-names per verb	32,767	32,767	32,767	511
ADD <i>identifier</i> . . .	N/L	N/L	N/L	N/L
ALTER <i>procedure-name</i> 1 TO <i>procedure-name</i> 2 . . .	4,194,303 ¹	4,194,303 ¹	4,194,303 ¹	64-KB ¹
CALL . . . BY CONTENT <i>identifier</i>	2,147,483,647	2,147,483,647	2,147,483,647	N/A
CALL <i>literal</i> . . .	4,194,303 ¹	4,194,303 ¹	4,194,303 ¹	N/L
CALL <i>identifier</i> or <i>literal</i> USING <i>identifier</i> or <i>literal</i> . . .	16,380	16,380	16,380	N/L
Active programs in run unit	32,767	32,767	32,767	32,767
Number of names called (DYN option)	N/L	N/L	N/L	64-K
CANCEL <i>identifier</i> or <i>literal</i> . . .	N/L	N/L	N/L	N/L
CLOSE <i>file-name</i> . . .	N/L	N/L	N/L	N/L
COMPUTE <i>identifier</i> . . .	N/L	N/L	N/L	N/L

Language element	Enterprise COBOL V5	Other Enterprise COBOL versions	IBM COBOL and VS COBOL II	OS/VS COBOL
DISPLAY <i>identifier</i> or <i>literal</i> . . .	N/L	N/L	N/L	N/L
DIVIDE <i>identifier</i> . . .	N/L	N/L	N/L	N/L
ENTRY USING <i>identifier</i> or <i>literal</i> . . .	N/L	N/L	N/L	N/L
EVALUATE . . . subjects	64	64	64	N/L
EVALUATE . . . WHEN clauses	256	256	256	N/L
GO <i>procedure-name</i> . . . DEPENDING	255	255	255	2031
INSPECT TALLYING and REPLACING clauses	N/L	N/L	N/L	15
MERGE <i>file-name</i> ASC or DES KEY . . .	N/L	N/L	N/L	12
Total merge key length	4092-B ⁷	4092-B ⁷	4092-B ⁷	256-B
MERGE USING <i>file-name</i> . . .	16 ⁸	16 ⁸	16 ⁸	16 ⁸
MOVE <i>identifier</i> or <i>literal</i> TO <i>literal</i> . . .	N/L	N/L	N/L	N/L
MULTIPLY <i>identifier</i> . . .	N/L	N/L	N/L	N/L
OPEN <i>file-name</i> . . .	N/L	N/L	N/L	N/L
PERFORM	4,194,303	4,194,303	4,194,303	64-K
SEARCH . . . WHEN . . .	N/L	N/L	N/L	N/L
SET <i>index</i> or <i>identifier</i> . . . TO	N/L	N/L	N/L	N/L
SET <i>index</i> . . . UP or DOWN	N/L	N/L	N/L	N/L
SORT <i>file-name</i> ASC or DES KEY	N/L	N/L	N/L	12
Total sort key length	4092-B ⁷	4092-B ⁷	4092-B ⁷	256-B
SORT USING <i>file-name</i> . . .	16 ⁸	16 ⁸	16 ⁸	16 ⁸
STRING <i>identifier</i> . . .	N/L	N/L	N/L	N/L
STRING DELIMITED <i>identifier</i> or <i>literal</i> . . .	N/L	N/L	N/L	N/L
UNSTRING DELIMITED <i>identifier</i> or <i>literal</i> . . .	N/L	255	255	15
UNSTRING INTO <i>identifier</i> or <i>literal</i> . . .	N/L	N/L	N/L	N/L
USE . . . ON <i>file-name</i> . . .	N/L	N/L	N/L	N/L

1. Items included in limit for procedure plus constant area.
2. No compiler limit, but VSAM limits it to 255 bytes.
3. QSAM limit.
4. Syntax checked, but has no effect on the execution of the program; there is no limit.
5. The compiler limit is shown, but QSAM limits it to 32,767 bytes.
6. For COBOL for OS/390 & VM V2R2 and later versions, 18 if ARITH(COMPAT) is in effect, or 31 if ARITH(EXTEND) is in effect.
7. For QSAM and VSAM, the limit is 4088 bytes if EQUALS is coded on the OPTION control statement.
8. SORT limit for QSAM and VSAM.
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 about using large block sizes, see the *Enterprise COBOL Programming Guide*.

Appendix G. Preventing file status 39 for QSAM files

To prevent file-status 39 for a QSAM file, ensure that there are no mismatches between the description of the file in your program and the attributes defined for the data 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:

File format	Requirement
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.
Format-F files	The record length specified in your program must exactly match the length attribute of the data set.
Format-U files	The maximum record length specified in your program must exactly match the length attribute of the data set.

Remember: Information in the JCL overrides information in the data set label.

For details about how record lengths are determined from the FD entry and record descriptions in your program, see the *Enterprise COBOL Programming Guide*.

Defining variable-length records

The easiest way to define variable-length records in your program is to use RECORD IS VARYING FROM integer-1 TO integer-2 in the FD entry and specify an appropriate value for integer-2. For example, assume that you have determined the length attribute of the 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

If 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
//OUTFILE DD  DSNAME=OUT171,UNIT=SYSDA,SPACE=(TRK,(50,5)),
//           DCB=(BLKSIZE=400)
```

/*

Enterprise 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 6. 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

If 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), use the following rules:

- 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  
| READING MODE IS V  
| RECORD VARYING 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

Note: This topic is for QSAM files only.

Enterprise COBOL dynamically allocates a file when all of the following conditions exist:

- The CBLQDA(ON) runtime option is in effect.
- A ddname for the file is not explicitly allocated.
- An environment variable of the same name is not set.
- The COBOL program opens the file to write to it.

When the file is opened, the attributes specified in your program will be used.

If CBLQDA(OFF) is in effect, an error will be generated.

Appendix H. TSO considerations

This appendix describes conversion considerations for programs running on TSO. It includes information about 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 you use 'Address TSO' (the default) or 'Address ATTCHMVS', both program parameters and Language Environment runtime options are processed. When using 'Address LINKMVS', runtime 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 I. Accessing JCL parameters

You can pass a parameter string from JCL to a COBOL program using the PARM= keyword of the EXEC statement. You can access these parameters either by standard COBOL coding, or by calling the CEE3PR2 Language Environment callable service.

Using COBOL coding

You must define the LINKAGE SECTION record (level-01) that is to receive the *user_parameter* data passed by the PARM string, taking into account the halfword length field that is inserted in front of the string by the system.

The program can test this field length for nonzero to verify that PARM-string data has in fact been passed. For example:

```
LINKAGE SECTION.  
01 PARMDATA.  
    05 STRINGLEN PIC 9(4) USAGE COMP.  
    05 STRINGPARM PIC X(80).  
PROCEDURE DIVISION USING PARMDATA.  
    IF STRINGLEN > 0 . . .
```

For more information, see Coding the LINKAGE SECTION in the *Enterprise COBOL Programming Guide*.

Using CEE3PR2

You must define parameters to the CEE3PR2 callable service, without the need to add parameters to your PROCEDURE DIVISION USING statement.

```
WORKING-STORAGE SECTION.  
01 PARMLN PIC S9(9) BINARY.  
01 PARMSTR.  
    02 STR1-LENGTH PIC S9(4) BINARY.  
    02 STR1-STRING.  
        03 STR1-CHAR PIC X  
            OCCURS 0 TO 256 TIMES  
            DEPENDING ON STR1-LENGTH.  
    . . .  
    CALL "CEE3PR2" USING PARMLN,PARMSTR, FC.
```

For more information about the CEE3PR2 callable service, see CEE3PR2 in the *Language Environment Programming Reference*.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Neither International Business Machines Corporation nor any of its affiliates assume any responsibility or liability in respect of any results obtained by implementing any recommendations contained in this article/document. Implementation of any such recommendations is entirely at the implementor's risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This information is intended to help you write programs using IBM Enterprise COBOL for z/OS. This Migration Guide documents General-Use Programming Interface and Associated Guidance Information provided for IBM Enterprise COBOL for z/OS. General-Use programming interfaces allow the customer to write programs that obtain the services of IBM Enterprise COBOL for z/OS.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms might or might not have the same meaning in other languages.

This glossary includes terms and definitions from the following publications:

- *ANSI INCITS 23-1985, Programming Languages - COBOL as amended by:*
 - *ANSI INCITS 23a-1989, Programming Languages - Intrinsic Function Module for COBOL,*
 - *ANSI INCITS 23b-1993, Programming Language - Correction Amendment for COBOL*
- *ANSI INCITS 172-2002 American National Standard Dictionary of Information Technology.*

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 a program.

above the 16-MB line

Storage above the so-called 16-MB line (or boundary) but below the 2-GB bar. This storage is addressable only in 31-bit mode (AMODE 31). Before IBM introduced the MVS/XA architecture in the 1980s, the virtual storage for a program was limited to 16 MB. Programs that have been compiled with 24-bit mode (AMODE 24) can address only 16 MB of space, as though they were kept under an imaginary storage line. Since VS COBOL II, a program that has AMODE 31 can address data above the 16-MB line.

*** 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 or collating sequence.

*** alphabetic character**

A letter or a space character.

*** alphanumeric character**

Any character in the computer's single-byte character set.

alphanumeric data item

A general reference to a data item that is described implicitly or explicitly as USAGE DISPLAY, and that has category alphanumeric, alphanumeric-edited, or numeric-edited.

alphanumeric-edited data item

A data item that is described by a PICTURE character string that contains at least one instance of the symbol A or X and at least one of the simple insertion symbols B, 0, or /. An alphanumeric-edited data item has USAGE DISPLAY.

*** 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.

AMODE

Provided by the linkage editor, the attribute of a load module that indicates the addressing mode in which the load module should be entered.

application

A collection of one or more routines cooperating to achieve particular objectives.

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**

(1) An expression used at the point of a call to specify a data item or aggregate to be passed to the called routine. (2) The data passed to a called routine at the point of call or the data received by a called routine.

*** 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 by one of the following operations:

1. A READ statement for a sequentially accessed file, when one of the following conditions is encountered:
 - No next logical record exists in the file.
 - The number of significant digits in the relative record number is larger than the size of the relative key data item.
 - An optional input file is not present.
2. A RETURN statement, when no next logical record exists for the associated sort or merge file.
3. A SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

B

basic document encoding

For an XML document, one of the following encoding categories that the XML parser determines by examining the first few bytes of the document:

- ASCII
- EBCDIC
- Unicode UTF-16, either big-endian or little-endian
- Other unsupported encoding
- No recognizable encoding

big-endian

The default format that the mainframe and the AIX[®] workstation use to store binary data and UTF-16 characters. In this format, the least significant byte of a binary data item is at the highest address and the least significant byte of a UTF-16 character is at 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 program, usually specified by a command or condition, where execution may be interrupted and control given to the workstation user or to a specified debug program.

Btrieve

A key-indexed record management system that allows applications to manage records by key value, sequential access method, or random access method. Enterprise COBOL supports COBOL sequential and indexed file I-O language through Btrieve.

buffer An area of storage into which data is read or from which it is written. Typically, buffers are used only for temporary storage.

built-in function

See "intrinsic function".

byte The basic unit of storage addressability. It has a length of 8 bits.

C**C language**

A high-level language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

C++ language

An object-oriented high-level language that evolved from the C language. C++ exploits the benefits of object-oriented technology such as code modularity, portability, and reuse.

callable services

A set of services that can be invoked by a Language Environment, featuring defined call interface, and usable by all programs sharing the Language Environment conventions.

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.

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.

CEEDUMP

A dump of the runtime environment for Language Environment and the member language libraries. Sections of the dump are selectively included, depending on options specified on the dump invocation. This is not a dump of the full address space, but a dump of storage and control blocks that Language Environment and its members control.

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**

A letter, digit, or other symbol that is used as part of the organization, control, or representative of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes.

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.

CICS Customer Information Control System.

CICS translator

A routine that accepts as input an application containing EXEC CICS commands and produces as output an equivalent application in which each CICS command has been translated into the language of the source.

*** 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 runtime object that represents a 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
?	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 directives are contained in the COBOL source program. Therefore, you can specify different suboptions of the directive within the source program by using multiple compiler-directive statements.

compiler options

Keywords that can be specified to control certain aspects of compilation. Compiler options can control the nature of the load module generated by the compiler, the types of printed output to be produced, the efficient use of the compiler, and the destination of error messages. See also compiler-time options.

compiler-time options

Keywords that can be specified to control certain aspects of compilation. Compiler options can control the nature of the load module generated by the compiler, the types of printed output to be produced, the efficient use of the compiler, and the destination of error messages.

*** 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 PROCEDURE 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.

*** 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 about 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

In COBOL, the part of a program that describes the files to be used in the program and the records contained within the files. It also describes any WORKING-STORAGE data items, LINKAGE SECTION data items, and LOCAL-STORAGE data items that are needed.

*** 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.

document type declaration

An XML element that contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or DTD.

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 or inserting editing characters.

*** editing character**

A single character or a fixed two-character combination belonging to the following set:

Character

Meaning

?	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
\$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	slant (virgule, slash)

element (text element)

One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type.

*** elementary item**

A data item that is described as not being further logically subdivided.

enclave

In Language Environment, an independent collection of routines, one of which is designated as the main routine and is invoked first. An enclave is roughly analogous to a program or run unit. An executable program..

***end class marker**

A combination of words, followed by a separator period, that indicates the end of a COBOL class definition. The end class marker is:

END CLASS class-name.

***end method marker**

A combination of words, followed by a separator period, that indicates the end of a COBOL method definition. The end method marker 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 marker**

A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program marker 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, 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

Synonym for run time.

execution-time environment

See "runtime 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**

Data that persists over the lifetime of an enclave and maintains last-used values whenever a routine within the enclave is reentered. Within an enclave consisting of a single load module, it is equivalent to any C data objects that have static storage duration, A FORTRAN common block, and COBOL EXTERNAL data.

*** 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 known 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 named collection of related data records that is stored and retrieved by an assigned name. Equivalent to an MVS data set.

*** 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 block

Block containing the addresses of I/O routines, information about how they were opened and closed, and a pointer to the file information block.

*** 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

A collection of files and their attributes. A file system provides a name space for file serial numbers referring to those files.

*** 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 routine that is invoked by coding its name in an expression. The routine passes a result back to the invoker through the routine name.

*** 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.

HLL High-level language.

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 bootstrap routine for COBOL/370 Release 1. It must be link-edited with any module that contains a COBOL/370 Release 1 program.

IGZCBSO

The bootstrap routine for COBOL for MVS & VM Release 2, COBOL for OS/390 & VM and Enterprise COBOL. It must be link-edited with any module that contains a COBOL for MVS & VM Release 2, COBOL for OS/390 & VM or Enterprise COBOL program.

IGZEBST

The bootstrap routine for VS COBOL II. It must be link-edited with any module that contains a VS COBOL II program.

ILC

InterLanguage Communication. 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.

*** 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.

IMS

Information Management System, IBM licensed product. IMS supports hierarchical databases, data communication, translation processing, and database backout and recovery.

*** 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.

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.

kernel The part of the component that contains programs for such tasks as I/O, management, and communication.

*** 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

Short form of z/OS Language Environment. A set of architectural constructs and interfaces that provides a common runtime environment and runtime services for C, C++, COBOL, FORTRAN and PL/I applications. It is required for programs compiled by Language Environment-conforming compilers and for Java applications.

Language Environment-conforming

Adhering to Language Environment's common interface conventions.

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.

link-edit

To create a loadable computer program by means of a linkage editor or binder.

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.

little-endian

Default format used by the PC to store binary data. In this format, the most significant digit is on the highest address. Compare with "big-endian".

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

*** 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

The first routine in an enclave to gain control from the invoker. In FORTRAN, a main program does not have a FUNCTION, SUBROUTINE, or BLOCK DATA statement as its first statement. It could have a PROGRAM statement as its first statement. Contrast with subprogram.

*** 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.
- 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*.
- MVS** Multiple Virtual Storage operating system.
- 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**
In COBOL, 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** Empty, having no meaning.
- * **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

A collection of one or more control sections produced by an assembler or compiler and used as input to the linkage editor or binder. Synonym for text deck or object 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.

*** 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 or external characteristics of the output medium.

*** page body**

That part of the logical page in which lines can be written 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 DIVISION, 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 DIVISION. The permissible paragraph headers in the IDENTIFICATION DIVISION are:

PROGRAM-ID. (Program ID DIVISION)
CLASS-ID. (Class ID DIVISION)
METHOD-ID. (Method ID 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

Data items that are received by a routine. The term used in certain other languages for the FORTRAN term dummy argument.

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**

In COBOL, a procedure is a paragraph or section that can only be performed from within the program. In PL/I, a named block of code that can be invoked externally, usually via a call..

*** 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 marker, a user-defined word or alphanumeric literal 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
?	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 runtime 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.		
runtime environment		
The environment in which a COBOL program executes.		
* run unit		
One or more object programs that are executed together. In Language Environment, a run unit is the equivalent of an enclave.		
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). A variable at the end of a statement.		
* 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 DIVISION. In the ENVIRONMENT and DATA DIVISION, 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.

*** 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 information 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 marker, if specified, or by the absence of additional source program lines. A source program contains a set of instructions written in a programming language that must be translated to machine language before the program can be run.

special character
A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign (hyphen)

*	asterisk
/	slant (forward slash)
=	equal sign
\$	currency sign
,	comma
;	semicolon
.	period (decimal point, full stop)
"	quotation mark
'	apostrophe
(left parenthesis
)	right parenthesis
>	greater than
<	less than
:	colon
_	underscore
	*> floating comment indicator

*** 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 the *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 The rules governing the structure of a programming language and the construction of a statement in a programming language.

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.

unpacked decimal format

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 1s (hex F). For example, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Synonymous with zoned decimal format.

*** 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.

volume

A certain portion of data, together with its data carrier, that can be handled conveniently as a unit. A data carrier mounted and demounted as a unit; for example, a reel of magnetic tape, a disk pack.

volume switch procedures

System specific procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

VSAM (Virtual Storage Access Method)

A high-performance mass storage access method. Three types of data organization are available: entry sequenced data sets (ESDS), key sequenced data sets (KSDS), and relative record data sets (RRDS). Their COBOL equivalents are, respectively: sequential, indexed, and relative organizations.

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.

X

XML Extensible Markup Language. A standard metalanguage for defining markup languages that was derived from and is a subset of SGML. XML omits the more complex and less-used parts of SGML and makes it much easier to write applications to handle document types, author and manage structured information, and transmit and share structured information across diverse computing systems. The use of XML does not require the robust applications and processing that is necessary for SGML. XML is developed under the auspices of the World Wide Web Consortium (W3C).

XML data

Data that is organized into a hierarchical structure with XML elements. The data definitions are defined in XML element type declarations.

XML declaration

XML text that specifies characteristics of the XML document such as the version of XML being used and the encoding of the document.

XML document

A data object that is well formed as defined by the W3C XML specification.

XML namespace

A mechanism, defined by the W3C XML Namespace specifications, that limits the scope of a collection of element names and attribute names. A uniquely chosen XML namespace ensures the unique identity of an element name or attribute name across multiple XML documents or multiple contexts within an XML document.

XML schema

A mechanism, defined by the W3C, for describing and constraining the structure and content of XML documents. An XML schema, which is itself expressed in XML, effectively defines a class of XML documents of a given type, for example, purchase orders.

year 2000 problem

The Year 2000 problem refers to the limitation of 2-digit year date fields that were used to save storage in the 1960s and 1970s. For example, it is not possible to compute the age of someone who is older than 100 years with 2-digit year date fields, and on 1/1/2000, the current date will not be greater than the previous day's date. Because so many applications and data have only 2-digit year data, they must all be changed before the year 2000 to avoid failure.

Z**zoned decimal format**

Synonym for unpacked decimal format.

zoned decimal item

See "external decimal item".

List of resources

IBM Enterprise COBOL for z/OS

You can find the following publications in the Enterprise COBOL for z/OS library:

- *Customization Guide*, SC14-7380
- *Language Reference*, SC14-7381
- *Programming Guide*, SC14-7382
- *Migration Guide*, GC14-7383
- *Program Directory*, GI11-9180
- *Licensed Program Specifications*, GI11-9181

Related publications

z/OS library publications

You can find the following publications in the z/OS Internet Library.

Run-Time Library Extensions

- *DWARF/ELF Extensions Library Reference*
- *Common Debug Architecture Library Reference*
- *Common Debug Architecture User's Guide*

z/Architecture

- *Principles of Operation*

z/OS DFSMS

- *Access Method Services for Catalogs*
- *Checkpoint/Restart*
- *Macro Instructions for Data Sets*
- *Using Data Sets*
- *Utilities*

z/OS DFSORT

- *Application Programming Guide*
- *Installation and Customization*

z/OS ISPF

- *Dialog Developer's Guide and Reference*
- *User's Guide Vol I*
- *User's Guide Vol II*

z/OS Language Environment

- *Concepts Guide*
- *Customization*

- *Debugging Guide*
- *Programming Guide*
- *Programming Reference*
- *Run-Time Messages*
- *Run-Time Application Migration Guide*
- *Writing Interlanguage Communication Applications*

z/OS MVS

- *JCL Reference*
- *JCL User's Guide*
- *Program Management: User's Guide and Reference*
- *System Commands*
- *z/OS Unicode Services User's Guide and Reference*
- *z/OS XML System Services User's Guide and Reference*

z/OS TSO/E

- *Command Reference*
- *Primer*
- *User's Guide*

z/OS UNIX System Services

- *Command Reference*
- *Programming: Assembler Callable Services Reference*
- *User's Guide*

z/OS XL C/C++

- *Programming Guide*
- *Run-Time Library Reference*

CICS Transaction Server for z/OS

You can find the following publications in the CICS Library:

- *Application Programming Guide*
- *Application Programming Reference*
- *Customization Guide*
- *External Interfaces Guide*

DB2 for z/OS

You can find the following publications in the DB2 Library:

- *Application Programming and SQL Guide*

- *Command Reference*
- *SQL Reference*

Debug Tool

You can find the following publications in the Debug Tool Library:

- *Reference and Messages*
- *User's Guide*

You can find the following publications by searching their publication numbers in the IBM Publications Center.

COBOL Report Writer Precompiler

- *Programmer's Manual*, SC26-4301

IMS

- *Application Programming API Reference*, SC18-9699
- *Application Programming Guide*, SC18-9698

Softcopy publications for z/OS

The following collection kit contains z/OS and related product publications:

- *z/OS CD Collection Kit*, SK3T-4269

Java

- *IBM SDK for Java - Tools Documentation*, publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp
- *The Java 2 Enterprise Edition Developer's Guide*, download.oracle.com/javaee/1.2.1/devguide/html/DevGuideTOC.html
- *Java 2 SDK, Standard Edition Documentation*, download.oracle.com/javase/1.4.2/docs/
- *The Java EE 5 Tutorial*, download.oracle.com/javaee/5/tutorial/doc/
- *The Java Language Specification, Third Edition*, by Gosling et al., java.sun.com/docs/books/jls/
- *The Java Native Interface*, download.oracle.com/javase/1.5.0/docs/guide/jni/
- *Java Technology Edition SDK User Guides*, www.ibm.com/developerworks/java/jdk/aix/service.html

Unicode and character representation

- *Unicode*, www.unicode.org/
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

XML

- *Extensible Markup Language (XML)*, www.w3.org/XML/
- *Namespaces in XML 1.0*, www.w3.org/TR/xml-names/
- *Namespaces in XML 1.1*, www.w3.org/TR/xml-names11/
- *XML specification*, www.w3.org/TR/xml/

Index

Special characters

/ (slash) in CURRENCY-SIGN clause
changed 76
* (asterisk) 64

Numerics

64-bit addressing 231

A

A in PICTURE clause 123
abbreviated combined relation conditions
parenthesis evaluation changed 64
abends
OCx, caused by unsupported
calls 259
U3504, caused by unsupported
calls 259
ACCEPT statement
keyword FROM requirements 64
system input devices for
mnemonic-name suboption 95
access JCL parameters
CEE3PR2 289
coding 289
LINKAGE SECTION 289
accessibility
keyboard navigation xxvii
of Enterprise COBOL xxvii
of this information xxvii
using of Enterprise COBOLEnterprise
COBOL xxvii
using z/OS xxvii
ACTUAL KEY clause 57
advantages of new compiler and run
time 6
AFTER phrase of PERFORM 82
ALPHABET clause 73, 109
ALPHABETIC class 72, 109
AMODE considerations 199
ANALYZE compiler option
not available in Enterprise
COBOL 145
applications
taking an inventory of (source) 27
APPLY CORE-INDEX clause 56
APPLY RECORD-OVERFLOW clause 57
APPLY REORG-CRITERIA clause 56
Area A, periods in 68, 97
ARITH compiler option
for converted IBM COBOL
programs 143
arithmetic accuracy 73
ASCII data set 284
ASRA abend failure symptom 259
assembler driver 260
assembler programs
call considerations
supported calls under CICS 259

assembler programs (*continued*)
call considerations (*continued*)
supported calls under
non-CICS 258
changing program mask 260
loading and BALRing COBOL 261
loading and deleting COBOL 261
paragraph name restrictions 74
ASSIGN ... FOR MULTIPLE REEL/UNIT
phrase 58
ASSIGN ... OR clause 58
ASSIGN clause 73
ASSIGN TO integer system-name
clause 58
assistive technologies xxvii
asterisk (*) 64

B

B in PICTURE clause 73, 123
BATCH compiler option 90
BDAM files 57
benefits of new compiler and run time 6
binder 229
binding 185
BLANK WHEN ZERO clause 64
BLL cells
automated conversion of 254
BUF compiler option 89
buffer size specification 89
BUFSIZE compiler option
for converted OS/VS COBOL
programs 89

C

CALL statement
changes for USING phrase 74
ON OVERFLOW,
CMPR2/NOCMPR2 110
callable services
CEETEST 203
calls
dynamic to alternate entry points 76
SOM services, to 140
supported
under CICS 259
under non-CICS 258
CCCA conversion tool
BDAM file conversion 57
brief description 54
detailed description 253
ISAM file conversion 56
reserved words 96, 106
CD FOR INITIAL INPUT 57
CEETEST callable service 203
changes to compiler, summary xv

CICS

call considerations
supported under Language
Environment 259
converting source programs
automatically (CCCA) 254
DATE special register 58
effect of TRUNC compiler
option 214
integrated translator 214
migrating separate translator to
integrated translator 214
OS/VS COBOL programs, support
for 43, 211
required compiler options
CICS 213
NODYNAM 213
RENT 213
CICS compiler option 11, 213, 215
CICS integrated translator 214
benefits of 214
CBL/PROCESS statements,
considerations for 214
comment lines, considerations
for 214
DFHCOMMAREA
considerations 214
migrating from separate
translator 214
TRUNC compiler option
considerations 215
CLOSE statement
DISP phrase unsupported 58
FOR REMOVAL phrase 65
POSITIONING phrase 58
CMPR2 116
CMPR2 compiler option
ALPHABET clause 109
ALPHABETIC class 109
CALL...ON OVERFLOW class 110
COPY statement 114
COPY...REPLACING statement 112
definition for 107
EXIT PROGRAM 117
file status codes 114
for converted VS COBOL II
programs 100
language differences from
NOCMPR2 108
not available with Enterprise
COBOL 12
PERFORM statement 119
PERFORM...VARYING...AFTER 121
PICTURE clause 123
PROGRAM COLLATING
SEQUENCE 126
READ INTO and RETURN
INTO 127
RECORD CONTAINS n
CHARACTERS 128
scaled integers and nonnumerics 111

- CMPR2 compiler option (*continued*)
 - SET...TO TRUE 129
 - SIZE ERROR on MULTIPLY and DIVIDE 130
 - UNSTRING statement 132
 - upgrading programs compiled with 107
 - upgrading VS COBOL II programs compiled with 93
 - UPSI switches 138
 - variable-length group moves 139
 - variable-length records 128
- COBOL
 - and Java
 - compatibility 232
 - COBOL 68 Standard 43
 - COBOL 85 Standard
 - interpretation changes 93
 - tools for converting source programs to 249
 - COBOL and CICS/VS Command Level Conversion Aid
 - detailed description 253
 - ISAM file conversion 56
 - COBOL applications
 - taking an inventory of (source) 27
 - COBOL for MVS & VM
 - upgrading to Enterprise COBOL 101
 - COBOL for OS/390 & VM
 - upgrading to Enterprise COBOL 101
 - COBOL/370
 - upgrading to Enterprise COBOL 101
 - CODE-SET clause, FS 39 283
 - comment lines
 - in VS COBOL II programs 94
 - Commonly asked questions 227
 - communication feature 57
 - comparing group to numeric packed-decimal item 65
 - compatibility
 - Java and COBOL 232
 - object-oriented syntax 232
 - compilation
 - Report Writer programs 55
 - compiler limits 279
 - compiler options
 - complete list 263
 - for compiling VS COBOL II programs 99
 - for converted OS/VS COBOL programs 89
 - for OS/VS COBOL, not supported 90
 - for SOM-based object-oriented COBOL, not supported 141
 - required for CICS integrated translator 215
 - upgrading from IBM COBOL 143
 - complexity ratings
 - conversion priorities relating to 30
 - conversion priority 28
 - conversion priority
 - complexity ratings relating to 30
 - conversion tools
 - CICS Application Migration Aid 26
 - CMPR2 compiler option 26

- conversion tools (*continued*)
 - COBOL Conversion Tool (CCCA) 26, 54, 253
 - Debug Tool Load Module Analyzer 256
 - Edge Portfolio Analyzer 256
 - FLAGMIG compiler option 26
 - FLAGMIG4 compiler option 26
 - MIGR compiler option 26, 54, 249
 - NOCOMPILE compiler option 26
 - Report Writer Precompiler 26, 255
- converting source
 - IBM COBOL programs, requiring 101
 - scenarios
 - Report Writer discarded 35
 - Report Writer retained 36
 - with CICS 33
 - without CICS or report writer 32
 - tasks when updating 37
- COPY statement 76
- COPY statement, using @, #, \$ 114
- COPY...REPLACING statement 112
- COUNT compiler option 90
- CURRENCY-SIGN clause 76
- CURRENT-DATE special register 58

D

- DATA DIVISION, two periods in a row 68
- data-name, unique compared to program-id 69
- DATA(24) compiler option
 - or converted OS/VS COBOL programs 89
- DATE FORMAT language elements support removed 174
- DATE special register 58
- DB2
 - coprocessor considerations 219
 - coprocessor integration 217
 - coprocessor migration 222
 - coprocessor, benefits of 217
 - separate precompiler 217
- debug information changes 163, 176, 191, 204
- Debug Tool 4, 204
- Debug Tool Load Module Analyzer 256
- debugging 163, 176, 191, 204
 - full screen mode 208
 - initiating the Debug Tool 203
 - remote mode 209
- DEBUGGING declarative 84
- decimal overflow, program mask and 260
- declaratives
 - changes to LABEL declarative support 180
 - debugging changes 84
 - GIVING phrase of ERROR 60
- DFHCOMMAREA
 - integrated CICS translator, considerations for 214
- DIAGTRUNC compiler option
 - for converted OS/VS COBOL programs 89

- disability xxvii
- DISP phrase of CLOSE 58
- DISPLAY statement 59
- DIVIDE statement 81, 130
- dynamic calls
 - CICS considerations
 - supported under Language Environment 259
 - placed to alternate entry points 76
 - supported under non-CICS under Language Environment 258

E

- Edge Portfolio Analyzer 256
- education
 - available for Enterprise COBOL 26
- enclave boundary with assembler programs 257
- ENDJOB compiler option 90
- Enterprise COBOL
 - advantages of 6
 - changes with 12
 - compiler options, complete list 263
 - compiler options, unsupported 99
 - high level overview 4
 - installing, documentation needed 25
 - JCL changes 185
 - logical record length 96
 - prolog format changes 91
 - reserved words, complete list 233
 - upgrading IBM COBOL programs to 14
 - upgrading VS COBOL II programs to 13
- Enterprise COBOL compiler limits 279
- Enterprise COBOL programs
 - existing applications, adding to 197
- Enterprise COBOL, upgrading OS/VS COBOL programs to 13
- ENTRY points 76
- ENVIRONMENT DIVISION, two periods in a row 68
- ERRCOUNT 230
- errors
 - subscripts out of range message 85
- evaluation changes in relation conditions 74
- EVENTS compiler option
 - not available in Enterprise COBOL 145
- EXAMINE statement 59
- EXEC CICS LINK
 - support under Language Environment 259
- EXEC CICS statement 215
- EXEC DLI statement 215
- EXHIBIT statement 59
- existing applications
 - adding Enterprise COBOL programs to 197
 - preventing file status 39 283
- EXIT PROGRAM statement 76
- differences between CMPR2 and NOCMPR2 117
- exponent underflow, program mask and 260

- exponentiation changes 73
- Extended Link Pack Area (ELPA) 215
- extensions, undocumented 64, 97
- External names, changed in Enterprise COBOL 141

F

- FAQ 227
- FD support in REDEFINES clause 70
- FDUMP compiler option
 - mapped to TEST 100
- file status 39
 - avoiding when processing new files 284
 - preventing for QSAM files 283
 - preventing for VSAM files 61
- FILE STATUS clause 76
- file status code
 - 39 98, 105, 159
- file status codes, CMPR2/
 - NOCMPR2 114
- FILE-CONTROL paragraph
 - FILE STATUS clause changed 76
 - FILE-LIMIT clause unsupported 60
- files
 - preventing file status 39 283
- fixed-length records, defining 284
- fixed-point overflow, program mask and 260
- FLAGMIG compiler option 12
 - definition for 107
 - not available with Enterprise COBOL 100
- FLAGSAA compiler option 100
- floating-point changes 73
- flow of control, ended 65, 117
- FOR REMOVAL phrase of CLOSE statement 65
- Format-x (F,S,U,V) files 283
- FROM, requirements with ACCEPT statement 64

G

- GENERATE statement 56
- GOBACK statement 65, 76
 - differences between CMPR2 and NOCMPR2 117

I

- IBM COBOL
 - upgrading source, requiring 14, 101
 - upgrading to Enterprise COBOL 101
- IDCAMS REPRO facility 57
- IDLGEN compiler option
 - not supported in Enterprise COBOL 141
- IF statement 79
- IGYPG3188 147
- IGYPG3189 147
- IGZ0005S 259
- IGZ0079S 259
- IGZ0193W 147
- IGZ0194W 147

- IGZERRE routine
 - for upgrading assembler driver 261
- ILBOSTP0
 - assembler driver, alternatives for 261
- index names
 - qualified 65
- INHERITS clause 140
- INITIATE statement 56
- INSPECT statement
 - EXAMINE statement 59
 - TRANSFORM statement 63
- installation
 - compiler, documentation needed 25
- INTDATE compiler option
 - for converted IBM COBOL programs 144
- integrated CICS translator 13, 214
 - required compiler options 215
- integrated DB2 coprocessor 217
- Integrated DB2 coprocessor 12
- integrated SQL coprocessor 217
- intermediate results changed 81
- inventory of applications 253
 - Debug Tool Load Module Analyzer 256
 - Edge's Portfolio Analyzer 256
 - for upgrading source to Enterprise COBOL 27
- INVOKE statement 141
- IS evaluation in relation conditions
 - changed 75, 81
- ISAM files 56

J

- Java
 - and COBOL
 - compatibility 232
- javac command
 - recompile for Java 232
- JUSTIFIED clause 79

K

- keyboard navigation xxvii

L

- LABEL RECORD clause 66
- LABEL RECORDS clause 60
- LANGLVL compiler option
 - unsupported 90
- LANGLVL(1) compiler option
 - /, =, and L characters 76
 - ACCEPT MESSAGE COUNT 57
 - combined abbreviated relational conditions 74
 - COPY statement with associated names 76
 - DELIMITED BY ALL 85
 - JUSTIFIED clause 79
 - NOT phrase 75
 - PERFORM statement 84
 - RESERVE clause 82
 - scaling change 79
 - SELECT OPTIONAL clause 84

- language elements
 - changed
 - OS/VS COBOL 72
 - SOM-based object-oriented COBOL 141
 - not supported
 - OS/VS COBOL 56, 58
 - SOM-based object-oriented COBOL 140
- Language Environment
 - advantages of 6
- Language Environment-conforming assembler programs 260
- LE's writable static area (WSA) 193
- LINE-COUNTER special register 56
- link editing 185
- Link Pack Area (LPA) 215
- link-editing 229
- LIST compiler option 91, 100
- LISTER features, unsupported 91
- load module analysis
 - Debug Tool Load Module Analyzer 256
 - Edge Portfolio Analyzer 256
- load modules
 - inventory of, using conversion tool 256
- LOAD/BALR calls supported under
 - Language Environment 258

M

- message IGZ0005S 259
- message IGZ0079S 259
- messages
 - MIGR, missing for RENAMES 71
- METAClass clause 141
- METHODS, changed in Enterprise COBOL 141
- METHODS, not supported in Enterprise COBOL 141
- MIGR compiler option
 - conversion tool 54, 249
 - message missing for RENAMES 71
- migrating CICS translator
 - from separate to integrated 214
- migrating from CMPR2 to NOCMPR2 107
- Migrating from
 - XMLPARSE(COMPAT) 150, 166
- migrating source
 - scenarios
 - Report Writer discarded 35
 - Report Writer retained 36
 - with CICS 33
 - without CICS or report writer 32
 - tasks when updating 37
- migration tools
 - COBOL and CICS/VS Conversion Aid (CCCA) 253
 - Debug Tool Load Module Analyzer 256
 - Edge Portfolio Analyzer 256
 - Report Writer Precompiler 255
- mnemonic-name of system input devices
 - in ACCEPT statement 95

MOVE ALL statement
to PIC 99 67

MOVE statement
CORRESPONDING changes 66
moving fullword binary items 66
multiple TO specification 67
scaling change 79
SET...TO TRUE 129
warning message for numeric
truncation 67

MULTIPLY statement 81, 130

N

national extension characters 114

new reserved words 158

NOCMPR2 116

NOCMPR2 compiler option
definition for 107
language differences from
CMPR2 108

NOCMPR2 programs
tools for converting source to 249

NOCOMPILE compiler option 90

NODYNAM compiler option 213, 215

NOMINAL KEY clause 56

nonnumerics, CMPR2/NOCMPR2 111

nonunique program-id names 69

NORENT compiler option
above the line support 11

NORENT static area 193

NORES compiler option 90
unsupported in Enterprise
COBOL 100

NOSTGOPT compiler option
for converted OS/VS COBOL
programs 89

NOT phrase 75

NOTE statement 61

NSYMBOL compiler option
for converted IBM COBOL
programs 144

numeric-edited, differences 69

NUMPROC compiler option
for converted OS/VS COBOL
programs 89

O

OBJECT COMPUTER paragraph 126

object module 229

object module, prolog format 91, 100

object-oriented COBOL
compatibility 232

object-oriented COBOL, SOM-based 12
compiler options not supported 141
language elements changed 141
language elements not
supported 140
not supported in Enterprise
COBOL 140

OBJECTS, changed in Enterprise
COBOL 141

OCCURS clause 67

OCCURS DEPENDING ON clause
changes in values for receiving
items 80

RECORD CONTAINS n
CHARACTERS 69
variable-length group moves 139

OCx abends 259

ODO objects, changes for variable-length
groups 94

ON SIZE ERROR phrase 81

ON statement 61

OPEN statement
COBOL 68 support dropped 61
REVERSED phrase changed 68

options
compiler
complete list 263
for IBM COBOL programs 143
for OS/VS COBOL programs 89
for VS COBOL II programs 99

ORGANIZATION clause 56, 57

OS/VS COBOL
ALPHABET-NAME clause
changed 73
arithmetic accuracy 73
ASSIGN clause changed 73
ASSIGN TO integer system-name
clause 58
CALL statement changed 74
compiler options, complete list 263
considerations when compiling 89
CURRENCY-SIGN clause changed 76
IF statement changed 79
intermediate results changed 81
JUSTIFIED clause 79
OCCURS DEPENDING ON
clause 80
ON SIZE ERROR phrase changed 81
PERFORM statement changes 82
PROGRAM COLLATING SEQUENCE
clause 82
READ statement changes 82
RERUN clause changes 82
RESERVE clause changes 82
reserved word list
complete list 233
RETURN statement changes 82
scaling changed 79
SEARCH statement changes 83
segmentation changes 84
SELECT OPTIONAL clause 84
SORT special register differences 84
source language debugging 84
subscripts out of range 85
undocumented extensions for 64
unsupported compiler options 90
UPSI switch evaluation changed 86
VALUE clause 86
VSAM files 77
WHEN-COMPILED 86
WRITE AFTER POSITIONING
statement 87

OS/VS COBOL compiler limits 279

OS/VS COBOL programs
CICS considerations
support for 211

OS/VS COBOL, upgrading source 13

OSDECK compiler option 91

OUTDD compiler option
for converted OS/VS COBOL
programs 89

P

PAGE-COUNTER special register 56

paragraph names
error for period missing in 69
requirements for Enterprise
COBOL 69, 74
restrictions for USING phrase 74

parameters
restrictions for paragraph names 74

parenthesis evaluation changed 75

PERFORM statement
difference between CMPR2 and
NOCMPR2 119
second UNTIL 68
VARYING/AFTER options 121
VARYING/AFTER phrases 82

periods
missing at end of SD, FD, or RD 68
missing on paragraph names 69
multiple in any division 68
requirements for Area A 68, 97

PGMNAME compiler option 99
for converted IBM COBOL
programs 144
for converted OS/VS COBOL
programs 89

PICTURE clause
B symbol in 73, 123
numeric-edited differences 69
use with VALUE clause 72

POSITIONING phrase of CLOSE 58

PPA4
how to find 192
layout 192

precedence of USE procedures 94

PROCEDURE DIVISION, two periods in
a row 68

program checks causing ASRA
abend 259

PROGRAM COLLATING SEQUENCE
clause
alphabet-name, implicit
comparisons 82
difference between CMPR2 and
NOCMPR2 126

program mask, programs that change
it 260

program names
compatibility 89, 99
requirements 69

program static area 193

prolog format 91, 100

Q

QSAM files
preventing files status 39 283
status key values 77

qualification - using the same phrase
repeatedly 69

qualified index names 65
QUEUE runtime option 57

R

READ statement
 implicit elementary MOVES 82
 INTO phrase, CMPR2/
 NOCMPR2 127
READY TRACE statement, not
 supported 61
RECEIVE statement 57
receiving fields, ODO objects 139
RECORD CONTAINS n CHARACTERS
 clause
 difference between CMPR2 and
 NOCMPR2 128
 when overridden 69
RECORD CONTAINS, fixed-length
 records 284
records, preventing FS 39 when
 defining 283
REDEFINES clause
 FD support dropped 70
 SD support dropped 70
reference modification 94
registers
 requirement for assembler
 programs 257
regression testing
 source considerations 38
relation condition
 coding changes 70
 evaluation changes 74
REMARKS paragraph 62
RENAMES clause 71
RENT compiler option 213, 215
RENT static area 193
REPLACE statement
 affecting EXEC CICS 215
REPLACE statement and comment
 lines 94
REPORT clause 56
report section 56
Report Writer
 conversion scenario discarding 35
 conversion scenario retaining 36
 conversion tool 55, 255
 language affected 56
Report Writer Precompiler 255
RERUN clause 82
RES compiler option 90, 100
RESERVE clause 82
reserved words
 comparison of 233
 comparison to VS COBOL II 96
RESET TRACE statement, not
 supported 61
return routine, assembler programs 257
RETURN statement
 implicit elementary MOVES 82
 INTO phrase, CMPR2/
 NOCMPR2 127
REVERSED phrase of OPEN
 statement 68
RMODE considerations 199

RRDS (relative-record data sets)
 simulating variable-length
 records 97, 104, 158
RTEREUS runtime option
 using with assembler drivers 260
runtime options
 HEAP 188
 NOCHECK 188
 NOSSRANGE 188
 SIMVRD 97, 104, 158
 STORAGE 188

S

scaled integers, CMPR2/NOCMPR2 111
SD support in REDEFINES clause 70
SEARCH ALL 106, 147
SEARCH statement 83
SEEK statement unsupported 57
segmentation 84
SELECT clause 84
sending fields, ODO objects 139
sequential files 77
SERVICE RELOAD statement
 automated conversion of 254
SET...TO TRUE, CMPR2/NOCMPR2 129
significance exceptions, program mask
 and 260
simplified TEST compiler option 161
SIMVRD runtime option 97, 104, 158
SIZE ERROR on MULTIPLY and
 DIVIDE 130
slash (/) in CURRENCY-SIGN clause
 changed 76
SOM-based object-oriented COBOL 12
 compiler options not available 141
 language elements changed 141
 language elements not
 supported 140
 not available with Enterprise
 COBOL 140
SORT special registers 84
source language conversion
 IBM tools 249
 inventory of applications 27
 tasks when updating 37
special registers
 CURRENT-DATE 58
 DATE 58
 LINE-COUNTER 56
 PAGE-COUNTER 56
 PRINT-SWITCH 56
 SORT differences 84
 TALLY 59
 TIME 63
 TIME-OF-DAY 63
 WHEN-COMPILED 86
SPECIAL-NAMES paragraph 76, 109
SPM instructions 260
SQL
 coprocessor integration 217
SQL statements
 DB2 coprocessor, handling 217
SSRANGE compiler option 85
STACK storage for work area 105
STANDARD LABEL statement 64

START statement
 support changed 63
 USING KEY clause unsupported 56,
 63
STATE compiler option 90
statement connectors, THEN
 unsupported 63
static CALL statement
 supported under Language
 Environment under CICS 259
 supported under Language
 Environment under non-CICS 258
status key
 QSAM files 77
 VSAM files 77
STOP RUN statement
 differences between CMPR2 and
 NOCMPR2 117
storage requirements
 compiler 25
subprograms
 dynamic calls to ENTRY points 76
subroutines, called by assembler
 driver 260
subscripts 85
SUPMAP compiler option 90
SVC LINK
 supported under Language
 Environment under non-CICS 258
 targeting assembler programs 257
SVC LOAD/BALR 261
SVC LOAD/DELETE 261
SXREF compiler option 90
SYMDMP compiler option 90
system input devices for mnemonic-name
 suboption in ACCEPT statement 95

T

TALLY special register 59
TERMINATE statement 56
terminating statements, required 65
TEST compiler option
 for converted VS COBOL II
 programs 99
testing
 regression, for source 38
THEN statement 63
TIME-OF-DAY special register 63
TRACK-AREA clause 56
TRACK-LIMIT clause 57
TRANSFORM statement
 unsupported 63
translator option
 XOPTS 214
translator, integrated CICS 214
TRUNC compiler option
 description 276
 for CICS applications 214, 215
 for converted IBM COBOL
 programs 144
 for converted OS/VS COBOL
 programs 90
 possible differences using
 TRUNC(OPT) 66

TYPECHK compiler option
not supported in Enterprise
COBOL 141

U

U3504 abends 259
undocumented extensions
for OS/VS COBOL 64
for VS COBOL II 97
uninitialized data sets 184
UNSTRING statement
coding not accepted 71
difference between CMPR2 and
NOCMPR2 132
multiple INTO phrases 72
upgrading
IBM COBOL programs 14
VS COBOL II programs 13
Upgrading programs from Enterprise
COBOL Version 3
Enterprise COBOL 147
Upgrading programs from Enterprise
COBOL Version 4
Enterprise COBOL 165
upgrading source
IBM COBOL programs,
requiring 101
IBM conversion tools 249
scenarios
Report Writer discarded 35
Report Writer retained 36
with CICS 33
without CICS or report writer 32
tasks when updating 37
upgrading, OS/VS COBOL programs 13
UPSI switches
difference between CMPR2 and
NOCMPR2 138
differences with condition-names 86
USE procedure
precedence in VS COBOL II 94
USE statement
BEFORE STANDARD LABEL 64
DEBUGGING declarative 84
GIVING phrase of ERROR
declarative 60
reporting declarative 56
Using REXX execs
processing parameter list
formats 287

V

VALUE clause
condition-name changes 86
use with PICTURE clause
changed 72
variable-length group moves 139
variable-length group, differences 94
variable-length records, defining 283
VARYING phrase of PERFORM
changed 82
VBREF compiler option 91
VBSUM compiler option 91

VCON
supported COBOL/assembler under
CICS 259
supported COBOL/assembler under
non-CICS 258
VS COBOL II
compiler options, complete list 263
reserved words, complete list 233
upgrading source 13
VS COBOL II compiler limits 279
VS COBOL II programs
reserved words, comparison 96
upgrading source programs 93
VSAM files
conversions 56
status key changes 77

W

WHEN-COMPILED special register 86
WORD(NO000) compiler option
for converted IBM COBOL
programs 145
WORKING-STORAGE
areas explanation 193
how to determine the area 194
WORKING-STORAGE data items 199
WORKING-STORAGE SECTION
how to find 192
in Enterprise COBOL V5 192
WRITE statement 87

X

XML PARSE compiler statements
XML parser 150
XML PARSE statements
XML parser 165
XMLSS suboption behavior 173
XOPTS translator option 214

Z

z/OS
commonly asked questions and
answers 231
Z's in PICTURE string 69



Product Number: 5655-W32

Printed in USA

GC14-7383-02

