Enterprise COBOL for z/OS

Compiler and Runtime Migration Guide

Version 3 Release 4

GC27-1409-05

Enterprise COBOL for z/OS

Compiler and Runtime Migration Guide

Version 3 Release 4

GC27-1409-05

Note!:

Before using this information and the product it supports, be sure to read the general information under Appendix N, "Notices," on page 327.

Sixth Edition (November 2006)

This edition applies to Version 3 Release 4 of IBM Enterprise COBOL for z/OS (program number 5655-G53), and to any subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product that you are using.

You can order publications online at www.ibm.com/shop/publications/order, or order by phone or fax. IBM Software Manufacturing Solutions takes publication orders between 8:30 a.m. and 7:00 p.m. EST (Eastern Standard Time). The phone number is (800)879-2755. The fax number is (800)445-9269.

You can also order publications through your IBM representative or the IBM branch office serving your locality.

© Copyright International Business Machines Corporation 1991, 2006. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document ix
Acknowledgement x
Using your documentation x
Enterprise COBOL for z/OS x
Language Environment element of z/OS, Version 1 Release 4 or later x
How to send your comments xi
Accessibility xi
Using assistive technologies xi
Keyboard navigation of the user interface . . . xi
Accessibility of this document xii

Summary of changes xiii
Changes to the Migration Guide 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 . . . xiii
Changes in GC27-1409-01, September 2002. . . xiii
Changes in GC27-1409-00, November 2001. . . xiv
Changes in GC26-4764-05, September 2000. . . xiv
Summary of changes to the COBOL compilers . . xiv
Version 3 Release 4, July 2005 xv
Changes in IBM Enterprise COBOL for z/OS, Version 3 Release 3 xvi
Changes in IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 2 xvii
Changes in IBM Enterprise COBOL for z/OS and OS/390, Version 3 Release 1 xvii
Changes in COBOL for OS/390 & VM, Version 2 Release 2 xviii
Changes in COBOL for OS/390 & VM, Version 2 Release 1 xix

Part 1. Overview 1

Chapter 1. Introducing the new compiler and run time 3
Product relationships—compiler, run time, debug . . 3
Comparison of COBOL compilers 4
Language Environment's runtime support for other programs 5
Advantages of the new compiler and run time . . . 6
Suggestions for incremental migration 11
Changes with the new compiler and run time . . . 12
CMPR2 compiler option not available 12
FLAGMIG compiler option not available . . . 12
ANALYZE compiler option not available . . . 12
SOM-based object-oriented COBOL not available 12
Integrated CICS translator available 13
General migration tasks 13
Planning your strategy 13
Moving to the Language Environment run time 13
Upgrading your source to Enterprise COBOL . . 14
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

Part 2. Migration strategies 21

Chapter 3. Planning the move to Language Environment 23
Preparing to move to the Language Environment runtime library 23
Installing Language Environment 23
Assessing storage requirements 23
Educating your programmers about Language Environment 26
Taking an inventory of your applications 26
Vendor tools, packages, and products. 27
COBOL applications 27
Assigning complexity ratings 28
Setting up conversion/no-conversion categories 30
Deciding how to phase Language Environment into production mode 30
Multilanguage conversion 30
Determining how applications will have access to the library 30
Setting up a regression testing procedure 33
Take performance measurements 34
Cutting over to production use 34

Chapter 4. Planning to upgrade source programs 35
Preparing to upgrade your source 35
Installing Enterprise COBOL 35
Assessing storage requirements 35
Deciding which conversion tools to use and install them 36
Educating your programmers on new compiler features. 36
Taking an inventory of your applications 37
Taking an inventory of vendor tools, packages, and products 37
Taking an inventory of COBOL applications . . 37
Prioritizing your applications 38
Setting up upgrade/no upgrade categories . . . 41
Setting up a conversion procedure. 41
Making application program updates. 45

© Copyright IBM Corp. 1991, 2006

Part 3. Moving existing applications to Language Environment 49

Chapter 5. Running existing applications under Language Environment 51

Set recommended default Language Environment runtime options 51
- Recommended runtime options for non-CICS applications 51
- Recommended runtime options for CICS applications 54

Invoking existing applications 56
- For non-CICS applications 56
- For CICS applications 57

Link-editing existing applications 57
Obtaining a system dump or a CICS transaction dump 58
- Method 1: Specify the TERMTHDACT runtime option 58
- Method 2: Specify an abnormal termination exit 58

Getting compatible abend behavior 60
Ensuring the compatibility of return-code values . . 60

Chapter 6. Moving from the OS/VS COBOL run time 61

Determining which programs require link-editing . 62
- Applications with COBOL programs compiled with RES 62
- Applications with COBOL programs compiled with NORES 62
- Applications with COBOL programs compiled with RES and NORES 62

Determining which programs require upgrading . . 63
- On CICS 63
- On non-CICS 63

Comparing runtime options and specification methods 64
- Specifying Language Environment runtime options 64
- Comparing OS/VS COBOL and Language Environment runtime options 66

Closing files in non-COBOL and OS/VS COBOL programs 66
- Other environments 66

Running in a reusable runtime environment . . . 67
- Using ILBOSTP0. 67

Managing dump services 68
- OS/VS COBOL symbolic dumps 68
- System storage dumps and CICS transaction dumps 68
- Language Environment formatted dumps . . . 68

Using ILBOABN0 to force an abend 69
Using SORT or MERGE in OS/VS COBOL programs 69
Understanding SYSOUT output changes. 70
- SYSOUT output with RECFM=FB 70
- OS/VS COBOL trace output sequence 70

Communicating with other languages 71
Additional CICS considerations. 71

Chapter 7. Moving from the VS COBOL II run time 73

Determining which programs require link-editing . 74
- Applications with COBOL programs compiled with RES 74
- Applications with COBOL programs compiled with NORES 74
- Programs that use ILC. 75
- Statically calling IGZCA2D or IGZCD2A . . . 75

Determining which programs require upgrading . . 75
- CICS 75
- Non-CICS 75

Comparing runtime options and specification methods 76
- Specifying Language Environment runtime options 76
- Specifying VS COBOL II runtime options . . . 78
- Comparing VS COBOL II and Language Environment options 79

Closing files in non-COBOL and OS/VS COBOL programs 81
- Other environments 81

Running in a reusable runtime environment . . . 81
- Precautions if establishing a reusable environment under IMS 82
- Using IGZERRE 83
- Using ILBOSTP0. 83
- Using RTEREUS 83

Managing messages, abend codes, and dump services. 84
- Runtime messages 84
- Timing of abend for runtime detected errors . . 85
- Abend codes 86
- Using CEEWUCHA 86
- Dump services 87

Using ILBOABN0 to force an abend 88
Using SORT or MERGE 89
- In OS/VS COBOL programs. 89
- In VS COBOL II subprograms 89

Understanding SYSOUT output changes. 90
- DISPLAY UPON SYSOUT and DD definitions . . 90
- SYSOUT output with RECFM=FB 90
- OS/VS COBOL trace output sequence 90

Communicating with other languages 91
- General ILC considerations 92
- COBOL and FORTRAN 92
- COBOL and PL/I 93
- COBOL and C/370 93

Initializing the runtime environment 94
- Existing applications using LIBKEEP 94
- Considerations for Language Environment preinitialization 95

Determining storage tuning changes 95
- Alternatives to IGZTUNE. 96
- Considerations for SPOUT output 97

Additional CICS considerations. 97
- Performance considerations 97
- SORT interface change. 97
- WORKING-STORAGE limits 97
- VS COBOL II NORENT programs 98
- IGZETUN or IGZEOPT and MSGFILE 98

CICS HANDLE commands and the CBLPSHPOP runtime option 98
DISPLAY statement 100
CLER transaction 101
Undocumented VS COBOL II extensions 101

Chapter 8. Link-editing applications with Language Environment 103
Applications comprised of NORES programs . . . 103
Implications of becoming RES-like 104
Applications comprised of RES programs 104

Chapter 9. Upgrading Language Environment release levels 105
Change in behavior for DATA(31) programs under OS/390, Version 2 Release 9, or later 105
Missing CEEDUMP for applications with assembler programs that use the DUMP macro under OS/390, Version 2 Release 8 106
Change in file handling for COBOL programs with RECORDING MODE U under OS/390, Version 2 Release 10 106
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 10 108
Calling between assembler and COBOL under OS/390, Version 2 Release 9 or later 108
Dynamic calls to assembler programs under Language Environment for z/OS, Version 1 Release 2 or later 109
Diagnosis of a COBOL program that has an FD entry for a VSAM file on CICS with Language Environment for z/OS, V1R2 or later 109
Referencing symbolic feedback tokens 110

Part 4. Upgrading source programs 111

Chapter 10. Upgrading OS/VS COBOL source programs 113
Comparing OS/VS COBOL to Enterprise COBOL 113
Language elements that require change—quick reference 113
Using conversion tools to convert programs to COBOL 85 Standard 117
COBOL Conversion Tool (CCCA) 117
OS/VS COBOL MIGR compiler option 117
CMPR2 and FLAGMIG compiler options . . . 117
Language elements that require other products for support 118
Report Writer 118
Language elements that are no longer implemented 119
ISAM file handling 119
BDAM file handling 120
Communication feature 121
Language elements that are not supported . . . 121
Undocumented OS/VS COBOL extensions that are not supported 128
Language elements that changed from OS/VS COBOL 136

Chapter 11. Compiling converted OS/VS COBOL programs 153
Key compiler options for converted programs . . 153
Unsupported OS/VS COBOL compiler options . . 154
Prolog format changes 155

Chapter 12. Upgrading VS COBOL II source programs 157
Determining which programs require upgrade before compiling with Enterprise COBOL 157
Upgrading VS COBOL II programs compiled with the CMPR2 compiler option 157
COBOL 85 Standard interpretation changes . . . 158
REPLACE and comment lines 158
Precedence of USE procedures. 158
Reference modification of a variable-length group receiver 159
ACCEPT statement 160
New reserved words 160
Undocumented VS COBOL II extensions 161

Chapter 13. Compiling VS COBOL II programs 163
Key compiler options for VS COBOL II programs 163
Compiling with Enterprise COBOL 163
Compiler options not supported in Enterprise COBOL 163
Prolog format changes 164

Chapter 14. Upgrading IBM COBOL source programs 165
Determining which programs require upgrade before you compile with Enterprise COBOL . . . 165
Upgrading SOM-based object-oriented (OO) COBOL programs 165
SOM-based object-oriented COBOL language elements that are not supported 166
Compiler options IDLGEN and TYPECHK . . 166
SOM-based object-oriented COBOL language elements that are changed 167
New reserved words in Enterprise COBOL . . . 167
Undocumented IBM COBOL extensions 168
Language elements changed from IBM COBOL . . 168
| Language Environment runtime considerations . . 168

Chapter 15. Compiling IBM COBOL programs 169
Default compiler options for IBM COBOL programs. 169
Key compiler options for IBM COBOL programs 169
Compiler options not available in Enterprise COBOL 171

~ **Part 5. Moving programs to**
~ **Enterprise COBOL 173**

Chapter 16. Migrating from CMPR2 to NOCMPR2 175

Upgrading programs compiled with the CMPR2 compiler option 175
- ALPHABET clause of the SPECIAL-NAMES paragraph 177
- ALPHABETIC class 177
- CALL . . . ON OVERFLOW 178
- Comparisons between scaled integers and nonnumerics 179
- COPY ... REPLACING statements using non-COBOL characters 180
- COPY statement using national extension characters 182
- File status codes 183
- Implicit EXIT PROGRAM 184
- PERFORM return mechanism 186
- PERFORM ... VARYING ... AFTER 188
- PICTURE clause with "A"s and "B"s 190
- PROGRAM COLLATING SEQUENCE 192
- READ INTO and RETURN INTO 193
- RECORD CONTAINS n CHARACTERS . . . 194
- Reserved words 195
- SET . . . TO TRUE 196
- SIZE ERROR on MULTIPLY and DIVIDE . . . 198
- UNSTRING operand evaluation 199
- UPSI switches 205
- Variable-length group moves 206

Chapter 17. CICS conversion considerations for COBOL source . . 209

Key compiler options for programs that run under CICS 209
Migrating from the separate CICS translator to the integrated translator 210
- Integrated CICS translator 211

Base addressability considerations for OS/VS COBOL programs 212
- SERVICE RELOAD statements 212
- LENGTH OF special register 213
- Programs using BLL cells 213
- Example 1: Receiving a communications area 214
- Example 2: Processing storage areas that exceed 4K 215
- Example 3: Accessing chained storage areas . . 216
- Example 4: Using the OCCURS DEPENDING ON clause 216

Chapter 18. Adding Enterprise COBOL programs to existing COBOL applications 219

Applications comprised of RES programs 219
- Adding Enterprise COBOL programs that use static CALL statements 219
- CALL statements on non-CICS 220
- CALL statements on CICS 221

Applications comprised of NORES programs . . . 221
- Behavior before link-editing with Language Environment 222
- Behavior after link-editing with Language Environment 222
- Link-edit override requirement 222

Multiple load module considerations 222
- OS/VS COBOL considerations. 222
- VS COBOL II considerations 224

AMODE and RMODE considerations 224
Runtime considerations 226
- ILBOSRV 226
- TGT (Task Global Table) and RSA (Register Save Area) conventions 226

~ Applications that contain assembler programs that
~ call COBOL 227

~ Chapter 19. Upgrading programs for ~ V3R4 SEARCH ALL 229

Part 6. Appendixes 233

Appendix A. Commonly asked questions and answers 235

Prerequisites. 235
Compatibility 235
Link-editing with Language Environment 237
Compiling with Enterprise COBOL 238
Language Environment services 239
Language Environment runtime options 240
Interlanguage communication 241
Subsystems 241
z/OS 243
Performance. 243
Service 243

Appendix B. COBOL reserved word comparison 245

Appendix C. Conversion tools for source programs 263

MIGR compiler option 263
- Language differences 263
- Statements supported with enhanced accuracy 264
- LANGLVL(1) statements not supported . . . 265
- LANGLVL(1) and LANGLVL(2) statements not supported 265

Other programs that aid conversion 266
- Report Writer for OS/2 and Windows 267
- WebSphere Studio Asset Analyzer 267
- COBOL and CICS/VS Command Level Conversion Aid (CCCA). 267
- CICS Application Migration Aid 269
- COBOL Report Writer Precompiler 269
- ~ Debug Tool Load Module Analyzer 270
- The Edge Portfolio Analyzer 270
- Vendor products 270

Appendix D. Applications with COBOL and assembler 271

Determining requirements for calling and called assembler programs 271
Calling assembler programs 271
Called assembler programs 272
SVC LINK and COBOL run-unit boundary . . 272
Runtime support for assembler COBOL calls on non-CICS 273
Runtime support for assembler COBOL calls on CICS 274
Converting programs that use ESTAE/ESPIE for condition handling 275
Error handling routines in existing programs 275
Converting programs that change the program mask 276
~ Calling assembler programs that expect a certain
~ program mask 277
Upgrading applications that use an assembler driver 278
Convert the assembler driver 278
Modify the assembler driver 278
Use an unmodified assembler driver 278
Invoking a COBOL program with an MVS ATTACH 279
Assembler loading and calling COBOL programs 280
| Assembler programs that load and BALR to MAIN
| COBOL programs 280
Assembler programs that load and delete COBOL programs 280
Freeing storage in subpools (z/OS and OS/390 only) 281
Invoking programs - AMODE requirements . . . 281

Appendix E. Debugging tool comparison 283
Debugging existing applications 283
Debugging migrated applications. 283
Applications with OS/VS COBOL programs . . 283
Applications with VS COBOL II programs. . . 283
Initiating Debug Tool 284
Command language comparison 285

Appendix F. Compiler option comparison 289

Appendix G. Compiler limit comparison 301

Appendix H. Preventing file status 39 for QSAM files 305
Processing existing files 305
Defining variable-length records 305
Defining fixed-length records 306
Converting existing files that do not match the COBOL record 306
Processing new files 306
Processing files dynamically created by COBOL 307

Appendix I. Overriding linkage editor defaults 309
When to override the default settings 309
How to override the defaults 309

Appendix J. Link-edit example 311

Appendix K. DB2 coprocessor integration 315
| Considerations when moving to the coprocessor 316

Appendix L. IMS considerations . . . 319
Unsupported VS COBOL II features 319
BLDL user exit unsupported 319
Compiler options relevant for programs run on IMS 319
Compiling and linking COBOL programs for running under IMS 319
ENDJOB/NOENDJOB compiler option requirements 320
Preloading requirements. 321
Last used state behavior under Language Environment 321
When programs remain in the last-used state 321
Recommended modules for preload 321
Enterprise COBOL programs 322
OS/VS COBOL programs 322
Condition handling using CBLTDLI on IMS . . . 322
Performance consideration when running OS/VS COBOL programs 322
Using a GTF trace to determine which modules are loaded. 322
DFSPCC20 modification unsupported 323

Appendix M. TSO considerations . . . 325
Using REXX execs 325

Appendix N. Notices 327
Programming interface information 327
Trademarks 328

List of resources 329
IBM Enterprise COBOL for z/OS. 329
Language Environment for z/OS 329
Language Environment for OS/390 329
Related publications 329

Glossary 331

Index 351

About this document

This book provides information to help you to move from a pre-Language Environment runtime library to IBM Language Environment for z/OS and to upgrade your source programs to Enterprise COBOL.

Terminology clarification

In this book, 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

In this book, 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

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	Version 3 Release 3	5655-G53
Enterprise COBOL	Version 3 Release 4	5655-G53

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

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

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

© Copyright IBM Corp. 1991, 2006

Acknowledgement

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

The information received from this previous conversion experience, as well as input from many experienced OS/VS COBOL and VS COBOL II IBM customers, aided in the development of this *Compiler and Runtime Migration Guide*. Without such assistance, this book would have been much more difficult to develop.

Using your documentation

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

Enterprise COBOL for z/OS

Table 2. The Enterprise COBOL for z/OS publications

Task	Information	Order number
Evaluate the product	*Fact Sheet*	GC27-1407
Understand warranty information	*Licensed Program Specifications*	GC27-1411
Install the compiler under z/OS	*Program Directory for z/OS*	GI10-8423
Understand product changes—upgrade source to Enterprise COBOL for z/OS and run time to Language Environment	*Compiler and Runtime Migration Guide*	GC27-1409
Customize Enterprise COBOL for z/OS	*Customization Guide*	GC27-1410
Prepare and test your programs and get details on compiler options	*Programming Guide*	SC27-1412
Get details on COBOL syntax and specifications of language elements	*Language Reference*	SC27-1408

Language Environment element of z/OS, Version 1 Release 4 or later

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

Task	Information	Order number
Evaluate the product	*Concepts Guide*	SA22-7567
Install Language Environment	*z/OS Program Directory*	GI10-0670
Understand Language Environment program models and concepts	*Programming Guide*	SA22-7561
Find syntax for Language Environment runtime options and callable services	*Programming Reference*	SA22-7562
Debug applications that run with Language Environment, get details on runtime messages, and diagnose problems with Language Environment	*Debugging Guide* *Run-Time Messages*	GA22-7560 SA22-7566
Migrate applications to Language Environment	*Run-Time Migration Guide*	GA22-7565
Develop interlanguage communication (ILC) applications	*Writing Interlanguage Applications*	SA22-7563

How to send your comments

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

- Fill out the Readers' Comment Form at the back of this book, and return it by mail or give it to an IBM representative. If the form has been removed, address your comments to:

  IBM Corporation
  H150/090
  555 Bailey Avenue
  San Jose, CA 95141-1003
  U.S.A.
- Fax your comments to this U.S. number: (800)426-7773.
- Use the Online Readers' Comment Form at www.ibm.com/software/awdtools/rcf/.

Be sure to include the name of the book, the publication number of the book, 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 a user who has a physical 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 enable users to:

- Use assistive technology products such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using only the keyboard.
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

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

Keyboard navigation of the user interface

Users can access z/OS user interfaces by using TSO/E or ISPF. For information about accessing TSO/E and ISPF interfaces, refer to the following publications:

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Volume I*

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 document

The English-language XHTML format of this document that will be provided in the IBM Problem Determination and Deployment Tools information center at http://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.

When you use JAWS for Windows, the links to accessible syntax diagrams might not work. Use IBM Home Page Reader to read the accessible syntax diagrams.

Summary of changes

Changes to the Migration Guide

This section lists the key changes that have been made to this book, the *Compiler and Runtime Migration Guide*, as a result of changes to COBOL compilers and runtime libraries.

The latest technical changes are marked by a revision bar in the left margin.

Changes in GC27-1409-05, November 2006

- Updated the documentation of differences between DB2 precompiler and coprocessor.
- Added compiler option SQLCCSID.

These changes are marked with a " ~ " in the left margin.

Changes in GC27-1409-04, March 2006

- Added to the documentation of differences between DB2 precompiler and coprocessor.
- Added a chapter on migrating SEARCH ALL statements to V3R4.

These changes are marked with a " ~ " in the left margin.

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.

These changes are marked with a " | " in the left margin.

Changes in GC27-1409-02, December 2003

- Applied service updates to the book.

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.

© Copyright IBM Corp. 1991, 2006

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 SOM-based OO COBOL syntax and programming model
- Added information on 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 10, "Upgrading OS/VS COBOL source programs," on page 113
- Added new reserved words
- Added information on migrating to the V2R2 compiler

Run time

- Added discussion 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 on upgrading Language Environment release levels

Miscellaneous maintenance and editorial changes have been made.

Summary of changes to the COBOL compilers

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

The latest technical changes are marked by a revision bar in the left margin.

Version 3 Release 4, July 2005

- 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.
- A number of 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.
- 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 *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 COBJVMINITOPTIONS 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, and features of Debug Tool Utilities and Advanced Functions:
  - 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 NOOPTIMZE 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 *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 jobjectArray 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 executing in the OS/390 UNIX System Services environment, with COBOL files able to reside in the HFS (hierarchical file system)
- 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.

Part 1. Overview

© Copyright IBM Corp. 1991, 2006

Chapter 1. Introducing the new compiler and run time

This chapter 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 book. This chapter includes information on the following:

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

Terminology clarification:

In this book, we use the term 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

In this book, 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 xiv for further details.

Product relationships—compiler, run time, 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 the COBOL 85 Standard. Some features such as the CMPR2 compiler option and SOM-based object-oriented COBOL syntax that IBM COBOL supported are no longer available with Enterprise COBOL.

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

© Copyright IBM Corp. 1991, 2006

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 Language Environment-conforming language programs including PL/I and C/C++.

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

Figure 1 shows the relationship between IBM's previous COBOL products and Enterprise COBOL, Language Environment, and Debug Tool.

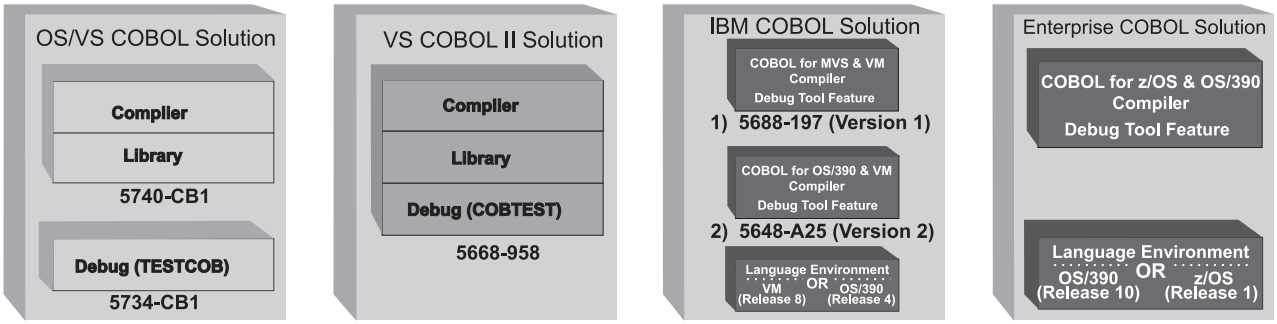


Figure 1. Product relationships—compiler, run time, and debug

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

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
		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, SAA flagging, Interactive debugging (full-screen mode)	COBOL 85 Standard, Structured programming, DBCS National language, Improved CICS interface, 31-bit addressing, Reentrancy, Fast Sort Optimizer, SAA flagging, Interactive debugging (full-screen mode)	COBOL 85 Standard, Structured programming, DBCS National language, Improved CICS interface, 31-bit addressing, Reentrancy, Fast Sort Optimizer, SAA flagging, Interactive debugging (full-screen mode)	COBOL 85 Standard, Structured programming, DBCS National language, Improved CICS interface, 31-bit addressing, Reentrancy, Fast Sort Optimizer, Interactive debugging (full-screen mode)
~ 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

| Language Environment, Version 1 Release 4 or later is available on z/OS for the Enterprise COBOL compiler.

Note: 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 other programs

Figure 2 on page 6 shows the programs that are able to run under Language Environment, which provides support for programs that currently run with the OS/VS COBOL and VS COBOL II libraries. Language Environment also provides support for other high-level languages.

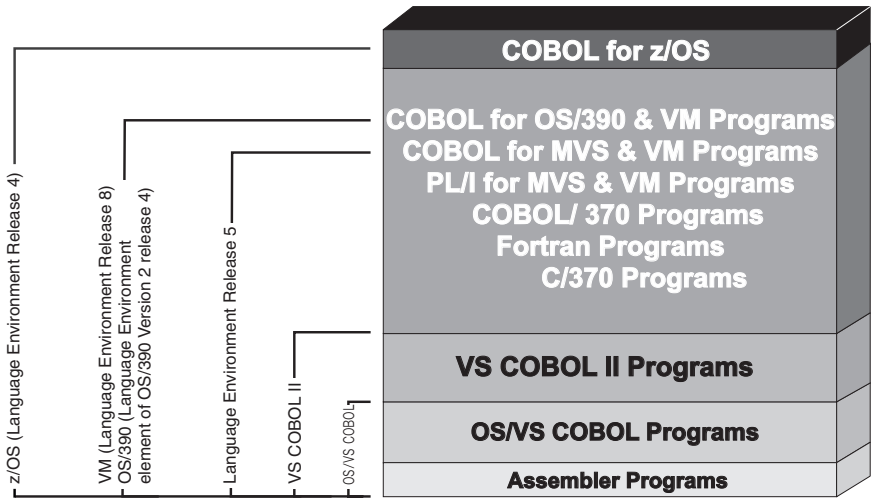


Figure 2. Language Environment's runtime support for other programs

Table 5 lists the version of Language Environment required for each operating system and the minimum operating system level required for Enterprise COBOL.

Table 5. Language Environment release levels

Host system	Language Environment release
z/OS, Version 1 Release 4 or later	Language Environment element of z/OS
Note: For a complete list of host versions and releases, see the *Licensed Program Specifications* for Language Environment.	

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 6 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 6. Advantages of Enterprise COBOL and Language Environment

		Advantage over		
Advantage	**Notes**	**OS/VS COBOL**	**VS II COBOL**	**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 *z/OS Support for Unicode*.	X	X	X
XML support	Enterprise COBOL provides new statements for parsing XML documents that allow programs to transform the XML content into COBOL data structures.	X	X	X

Table 6. Advantages of Enterprise COBOL and Language Environment (continued)

		Advantage over		
Advantage	**Notes**	**OS/VS COBOL**	**VS II COBOL**	**IBM COBOL**
Integrated CICS translator	The Enterprise COBOL compiler handles both native COBOL and embedded CICS statements in the source program.	X	X	*
Usability enhancements	These enhancements include: • 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.	X	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	
Strategic IBM compiler	Any future performance improvements or language enhancements for the z/OS platform, will only be available with Enterprise COBOL and Language Environment.	X	X	X
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	
Cost savings	If your shop uses multiple languages, you could see a cost savings by replacing multiple language run times with the single Language Environment run time. Talk with your IBM representative to evaluate the potential cost savings based on the number of current licenses and languages used by your shop.	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	

Table 6. Advantages of Enterprise COBOL and Language Environment (continued)

		Advantage over		
Advantage	**Notes**	**OS/VS COBOL**	**VS II COBOL**	**IBM COBOL**
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.	X	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, on 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: • Inline PERFORM statements • The CONTINUE place-holder statement • The EVALUATE statement • Explicit scope terminators (for example: END-IF, END-PERFORM, END-READ)	X		
COBOL 85 Standard conformance	Support for the COBOL 85 Standard	X		
	Support for Amendment 1 (Intrinsic Functions Module) of the COBOL 85 Standard	X	X	
Subsystem support	Improved support for IMS, ISPF, DFSORT, DB2	X		
Improved DB2 function	Enterprise COBOL includes support for DB2 stored procedures.	X	X	
	Support for the DB2 coprocessor (available in DB2, Version 7)	X	X	*

Table 6. Advantages of Enterprise COBOL and Language Environment (continued)

		Advantage over		
Advantage	**Notes**	**OS/VS COBOL**	**VS II COBOL**	**IBM COBOL**
Improved CICS interface	Enterprise COBOL includes CALL statement support (for faster CICS performance than when using EXEC CICS LINK) and eliminates the need for BLL cells. See "Base addressability considerations for OS/VS COBOL programs" on page 212.	X		
	Increased WORKING-STORAGE space for DATA(24) and DATA(31) programs. For DATA(31), the limit is 128M. For DATA(24), the limit is the available space below the 16-MB line.	X	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: • 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 WebSphere Developer for zSeries, 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: • Dynamic Debug feature which allows COBOL for OS/390 programs compiled without hooks to be debugged. • The SEPARATE suboption added to the TEST compiler option. Using this option produces a separate debug file that Debug Tool uses when debugging COBOL programs.	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 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 6. Advantages of Enterprise COBOL and Language Environment (continued)

		Advantage over		
Advantage	**Notes**	**OS/VS COBOL**	**VS II COBOL**	**IBM COBOL**
Compiler options	The following compiler options are available to Enterprise COBOL programs only: • 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
	The following compiler options are available to COBOL for OS/390 & VM and later programs only: • DLL - enables the compiler to generate an object module that is enabled for Dynamic Link Library (DLL) support. • EXPORTALL - instructs the compiler to automatically export certain symbols when the object deck is link-edited to form a DLL.	X	X	
	The following compiler options are available to COBOL for MVS & VM and later programs: • CURRENCY - allows you to define a default currency symbol for COBOL programs. • OPTIMIZE(FULL) - OPTIMIZE with the new suboption of FULL optimizes object programs and provides improved 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	

Table 6. Advantages of Enterprise COBOL and Language Environment (continued)

		Advantage over		
Advantage	**Notes**	**OS/VS COBOL**	**VS II COBOL**	**IBM COBOL**
	Enterprise COBOL provides, as does IBM COBOL and VS COBOL II, compiler options that give you added control over compiler output, such as: • Object code generation • Compiler usage of virtual storage • Listings, maps, and diagnostics • Runtime debugging information • Customized reserved word lists • Processing COPY or BASIS statements • Text of error messages • Language of error messages	X		
Note: * 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.				

Suggestions for incremental migration

If for any reason your shop is unable to migrate to Enterprise COBOL and Language Environment, you can still take incremental steps to prepare for migration when these obstacles no longer apply. For example:

- Evaluate the effort to move to Language Environment.
- Develop a migration plan and long-range schedule.
- Convert your macro-level CICS programs to command-level programs. (For guidance information, see "CICS Application Migration Aid" on page 269.)
- Code applications based on Enterprise COBOL and Language Environment requirements to ease a future migration. For example, specify the RES compiler option instead of the NORES compiler option; all Enterprise COBOL programs are RES. A "RES-migration" is easier than a "NORES-migration." Link-editing NORES applications with Language Environment can result in different behavior, depending on how the programs in the application are coded. For details, see Chapter 8, "Link-editing applications with Language Environment," on page 103.
- Determine which applications contain ILC between C and COBOL. Verify that the COBOL and C used are supported by Language Environment and make the appropriate changes as described in the *C Migration Guide*. For details, see "COBOL and C/370" on page 93.
- Determine which applications contain ILC between PL/I and VS COBOL II. Verify that the COBOL and PL/I used are supported by Language Environment, and link-edit the PL/I programs with the PL/I migration tool. For details, see "COBOL and PL/I" on page 93.

  The PL/I migration tool allows you to link-edit your PL/I programs gradually, while still running on the PL/I run time. If you link-edit with the migration tool, you do not have to link-edit before running under Language Environment.
- Convert all COBOL source code to COBOL 85 Standard.

Changes with the new compiler and run time

With Enterprise COBOL, you will find that existing COBOL applications are affected by several areas such as the removal of compiler options, different default compiler options, unsupported SOM-based OO COBOL, and an integrated CICS translator. Following 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 (1985 COBOL Standard) in order to compile them with Enterprise COBOL.

For additional details, see:
- Chapter 10, “Upgrading OS/VS COBOL source programs,” on page 113
- Chapter 12, “Upgrading VS COBOL II source programs,” on page 157
- Chapter 14, “Upgrading IBM COBOL source programs,” on page 165

FLAGMIG compiler option not available

Enterprise COBOL does not provide the FLAGMIG compiler option. FLAGMIG requires CMPR2, which is not provided with Enterprise COBOL. 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, or use CCCA.

For additional details, see:
- Chapter 10, “Upgrading OS/VS COBOL source programs,” on page 113
- Chapter 12, “Upgrading VS COBOL II source programs,” on page 157
- Chapter 14, “Upgrading IBM COBOL source programs,” on page 165

ANALYZE compiler option not available

Enterprise COBOL does not support the ANALYZE compiler option. With IBM COBOL, specification of the ANALYZE option allowed a syntax check of programs that contained SQL or CICS statements. Specifying both the ANALYZE and ADATA options created a SYSADATA file which could be analyzed by program understanding tools. However, the program understanding tools are no longer available in VisualAge COBOL, and Enterprise COBOL now supports SQL and CICS statements through the SQL and CICS compiler options. The ADATA compiler option can create a similar SYSADATA file without the need for further analysis by program understanding tools.

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 includes the removal of the compiler options TYPECHK and IDLGEN because they require SOM-based OO COBOL 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 165

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 if you have CICS Transaction Server, Version 2 installed.

The CICS compiler option must be specified to enable the CICS translator to process a COBOL source programs that contains CICS statements. For additional details and compatibility considerations, see:

- Chapter 17, "CICS conversion considerations for COBOL source," on page 209

General migration tasks

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

- Planning your strategy
- Moving to the Language Environment runtime library
- Upgrading your source to Enterprise COBOL
- Adding Enterprise COBOL programs to existing applications

Planning your strategy

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

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

- Chapter 3, "Planning the move to Language Environment," on page 23
- Chapter 4, "Planning to upgrade source programs," on page 35

Moving to the Language Environment run time

You can run existing load modules under Language Environment and receive the same results as with pre-Language Environment libraries. For important compatibility information, see Chapter 5, "Running existing applications under Language Environment," on page 51.

For information on moving applications that are running under the OS/VS COBOL run time, see Chapter 6, "Moving from the OS/VS COBOL run time," on page 61.

For information on moving applications that are running under the VS COBOL II run time, see Chapter 7, "Moving from the VS COBOL II run time," on page 73. (Because you might have been running OS/VS COBOL programs under the VS COBOL II run time, this chapter duplicates the information from Chapter 5 that is applicable to OS/VS COBOL programs running under the VS COBOL II run time.)

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

- "Determining which programs require link-editing" on page 62 for programs running under the OS/VS COBOL run time

- "Determining which programs require link-editing" on page 74 for programs running under the VS COBOL II run time

Link-editing programs with Language Environment can result in different behavior, see Chapter 8, "Link-editing applications with Language Environment," on page 103 for details.

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

- "Determining which programs require upgrading" on page 63 for programs running under the OS/VS COBOL run time
- "Determining which programs require upgrading" on page 75 for programs running under the VS COBOL II run time
- "Determining which programs require upgrade before you compile with Enterprise COBOL" on page 165 for IBM COBOL programs.

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.

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 "Using conversion tools to convert programs to COBOL 85 Standard" on page 117.

VS COBOL II

From a conversion standpoint, the only language difference between VS COBOL II, Release 4 and Enterprise COBOL is the addition of new reserved words. A complete list of reserved words, including those reserved for object-oriented COBOL is included in Appendix B, "COBOL reserved word comparison," on page 245.

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 12, "Upgrading VS COBOL II source programs," on page 157.

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 no longer 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 the COBOL 85 Standard. Details of language differences between CMPR2 and NOCMPR2 are included in Chapter 16, "Migrating from CMPR2 to NOCMPR2," on page 175.

For details on the conversion tools available to upgrade source programs, see Appendix C, "Conversion tools for source programs," on page 263.

IBM COBOL

Your IBM COBOL programs will compile without change using the Enterprise COBOL compiler unless you have one or more of the following:

- 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

For details, see Chapter 14, “Upgrading IBM COBOL source programs,” 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.

When adding Enterprise COBOL programs to existing applications, you need to be aware of the effects of link-editing with Language Environment, the restrictions of running programs, acquiring WORKING-STORAGE, or both above or below the 16-MB line, the effect of compiler option changes, reserved word changes, and other behavior differences with Enterprise COBOL. For details, see Chapter 18, “Adding Enterprise COBOL programs to existing COBOL applications,” on page 219.

Chapter 2. Do I need to recompile?

Ideally, programs should be compiled with a supported compiler (recommended is IBM Enterprise COBOL for z/OS) and run with a supported runtime library (Language Environment). You can migrate programs gradually, in two stages:
- Stage 1: Runtime migration
- Stage 2: Compiler migration (optional, depending on the need to recompile)

For most programs, you *do not* need to recompile to ensure that they continue working properly or have service support. For details on programs that must be upgraded, see:
- "Determining which programs require upgrading" on page 63 for OS/VS COBOL programs running with the OS/VS COBOL run time
- "Determining which programs require upgrading" on page 75 for OS/VS COBOL and VS COBOL II programs running with the VS COBOL II run time
- "Determining which programs require upgrade before you compile with Enterprise COBOL" on page 165 for IBM COBOL programs

The remainder of this chapter explains when and why you might want to migrate your applications (runtime or source). It includes the following topics:
- Migration basics
- Service support for OS/VS COBOL and VS COBOL II programs

Migration basics

The migration process involves runtime migration (moving your applications to a new run time) 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 move your run time and source concurrently.

For more details on 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.

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.

© Copyright IBM Corp. 1991, 2006

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.

For additional information on moving your run time to Language Environment, see:
- Chapter 5, "Running existing applications under Language Environment," on page 51
- Chapter 6, "Moving from the OS/VS COBOL run time," on page 61
- Chapter 7, "Moving from the VS COBOL II run time," on page 73

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 usually consists of upgrading the source language level that is used (such as from the COBOL 74 Standard supported by OS/VS COBOL to the 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 263.

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

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 exception: OS/VS COBOL programs running on CICS Transaction Server Version 3.

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 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.

In CICS TS (Transaction Server), Version 2 Release 2 or later, you can no longer use the CICS Translator for OS/VS COBOL programs. As soon as possible, upgrade to Enterprise COBOL any OS/VS COBOL programs that need to change to run under CICS, Version 2 Release 2 or later.

Changing OS/VS COBOL programs

Although the OS/VS COBOL compiler is no longer supported, the programs that were generated by it are supported if they are running under Language Environment. Once you have moved your run time to Language Environment, you

can run your source code through a source conversion tool, such as the COBOL and CICS Conversion Aid (CCCA) and then compile using the Enterprise COBOL compiler.

For more information on CCCA, see Appendix C, “Conversion tools for source programs,” on page 263.

Part 2. Migration strategies

© Copyright IBM Corp. 1991, 2006

Chapter 3. Planning the move to Language Environment

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

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

Important

On z/OS, Enterprise COBOL programs can run with the Language Environment element of z/OS Version 1 Release 4 or later.

Preparing to move to the Language Environment runtime library

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

- Installing Language Environment.
- Educating your programmers about Language Environment.
- Assessing storage requirements.

Installing Language Environment

To install z/OS, including the Language Environment element, refer to either the *z/OS Program Directory* or consult your *ServerPac: Installing Your Order*.

Important: To ensure that the Language Environment runtime results are compatible with pre-Language Environment results, you might need to change the default runtime options. For a list of recommended runtime options, see "Set recommended default Language Environment runtime options" on page 51.

Assessing storage requirements

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

DASD storage requirements

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

To determine the amount of DASD storage required by Language Environment, see *z/OS Program Directory*

Virtual storage requirements

Virtual storage requirements for running COBOL programs with Language Environment will increase over both the OS/VS COBOL run time and the VS

© Copyright IBM Corp. 1991, 2006

COBOL II run time. For both CICS and non-CICS applications, the amount of increase depends on many factors, such as:

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

  Note: You can use the information generated by the Language Environment RPTSTG runtime option to help tune your storage options. For details, see the *Language Environment Programming Reference*.
- The value used for the Language Environment runtime option ALL31.
- Which runtime routines are in the LPA (link pack area) or the ELPA (extended link pack area)
- Additionally, when moving from the VS COBOL II run time:
  - VS COBOL II runtime options specification
  - If the COBPACKs have been modified to run above the 16-MB line.

Important

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

Virtual storage information for non-CICS: Table 7 shows how much virtual storage is used when a simple VS COBOL II program compiled with RES was run on z/OS. In each case, the runtime library routines were acquired from STEPLIB:

Table 7. Virtual storage used by a VS COBOL II RES program on non-CICS

Runtime library	Modifications from IBM default	Below 16-MB virtual storage	Above 16-MB virtual storage
VS COBOL II, Release 4	None	276K	4K
VS COBOL II, Release 4	COBPACKs modified to run above the 16-MB line	32K	192K
Language Environment element of z/OS, Version 1 Release 6	None	88K	2636K
Language Environment element of z/OS, Version 1 Release 6	Runtime options ALL31(OFF) and STACK(BELOW)	280K	2504K

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

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

The most important factors include:

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

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

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

The following table shows storage usage data for the VS COBOL II run time and Language Environment run time on CICS TS Version 2 Release 2. The data was collected by running a simple transaction where one VS COBOL II program does an EXEC CICS LINK to another VS COBOL II program with auxiliary trace turned on. The amount of storage used was determined by looking at the GETMAIN trace entries in the auxiliary trace output.

Table 8. Storage allocation for CICS applications

Storage allocated	VS COBOL II run time	Language Environment run time
Per transaction	2040 bytes Storage is below the 16-MB line if the transaction is defined with TASKDATALOC(BELOW). Storage is above the 16-MB line if the transaction is defined with TASKDATALOC(ANY). Fixed size used for control blocks.	14312 bytes Storage is below the 16-MB line if the transaction is defined with TASKDATALOC(BELOW). Storage is above the 16-MB line if the transaction is defined with TASKDATALOC(ANY). Under CICS TS, Version 1 Release 3 with APAR PN85951 applied, or CICS Transaction Server, the storage will always be allocated above the 16-MB line. Fixed size used for control blocks.

Table 8. Storage allocation for CICS applications (continued)

Storage allocated	VS COBOL II run time	Language Environment run time
Per run unit	740 bytes below the 16-MB line (400 bytes for GETMAIN and 340 bytes for control blocks) On the first run unit, above and below the 16-MB line heap storage is allocated. This storage is used by any new run units created during the transaction. Additional storage is allocated as needed. When using the IBM-supplied defaults, 8168 bytes of below-the-line storage and 16352 bytes of above-the-line storage is allocated when the first run unit starts.	
Per run unit with ALL31(ON)		33336 bytes above the 16-MB line. This includes storage for control blocks, plus storage for STACK, LIBSTACK, HEAP, and ANYHEAP. If any below the 16-MB line heap is needed, it is allocated on demand.
Per run unit with ALL31 (OFF) and STACK (4K,4K,BELOW, KEEP)		18000 bytes above the 16-MB line. This includes storage for control blocks, plus storage for HEAP and ANYHEAP. 19672 bytes below the 16-MB line. This includes storage for control blocks, plus storage for LIBSTACK, STACK, and BELOWHEAP.

Note: This scenario was run on Language Environment, Version 1 Release 6 using the IBM-supplied values for runtime options: STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP.

Educating your programmers about Language Environment

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

Once your programmers are familiar with Language Environment, they can better prepare for the move to Language Environment. For example, they can assist in taking an inventory of applications. They can also code according to the suggestions listed in “Suggestions for incremental migration” on page 11.

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

Taking an inventory of your applications

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

- Vendor tools, packages, and products
- COBOL applications

The Debug Tool Load Module Analyzer can determine the language translator for each object in your load modules. See "Debug Tool Load Module Analyzer" on page 270 for more information.

The Edge Portfolio Analyzer can aid in taking an inventory of your existing load modules. See "The Edge Portfolio Analyzer" on page 270 for more information.

The WebSphere Studio Asset Analyzer for z/OS can aid by analyzing the impact of a code change for an application. See "WebSphere Studio Asset Analyzer" on page 267 for more information.

Vendor tools, packages, and products

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

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

For a list of vendor products that are enabled for Language Environment, see *Language Environment Run-Time Migration Guide*.

COBOL applications

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

For moving your applications to Language Environment:

- Which programs have been compiled with VS COBOL II and which with OS/VS COBOL
- Which programs have been compiled with RES and which with NORES
- Runtime options used (and how specified), specifically:
  - MIXRES (VS COBOL II)
  - RTEREUS (VS COBOL II)
  - LIBKEEP (VS COBOL II)
  - FLOW (OS/VS COBOL)
- Which COBOL programs call or are called by assembler programs
- Which COBOL programs use interlanguage communication (PL/I, C, or FORTRAN)
- Which COBOL programs are used under CICS, IMS, DB2, or other subsystems
- Control statements used
- Frequency and types of abends

For regression testing:

- Test cases required and available

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

Assigning complexity ratings

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

Each program attribute is assigned a complexity rating, as defined in the following table:

Table 9. Complexity ratings for program attributes

Complexity rating	Definition
0-2	Requires minimum testing, or Runs under Language Environment without change and without link-editing with Language Environment
3-6	Requires moderate testing, or Requires moderate coordination, or Requires link-editing with Language Environment
7-10	Requires moderate to high degree of testing, or Requires moderate to high degree of coordination, or Requires rewrite of module, or Does not run under Language Environment

When moving from the OS/VS COBOL run time

Table 10 shows estimated complexity ratings for conversions of specific program attributes:

Table 10. Complexity ratings for programs running under the OS/VS COBOL run time

Program attribute	Complexity rating
Compiled with NORES and link-edited with OS/VS COBOL	0
CICS online	1
Link-editing programs using ILBOSTP0 with the assembler driver	3
IMS online	5
Mixed RES and NORES programs	5
ISPF program	7
Called by assembler routine with a LINK SVC	7
CICS (if using dynamic CALL statements from OS/VS COBOL programs)	7
Assembler programs that do not follow normal save area conventions. For details, see "Determining requirements for calling and called assembler programs" on page 271.	8
ILC between OS/VS COBOL programs and PL/I programs	8
ILC between OS/VS COBOL programs and FORTRAN programs	8
Multiple load module application, where the main module has OS/VS COBOL NORES programs and no Enterprise COBOL or VS COBOL II programs are included in the main module.	10

Table 10. Complexity ratings for programs running under the OS/VS COBOL run time (continued)

Program attribute	Complexity rating
Uses QUEUE runtime option	10
Assembler programs that issue a STAE or SPIE	10
Calls assembler routines that do not have valid 31-bit addresses in Register 13 for save area changes.	10
Assembler programs that are coded based on the internals of the ILBO routines	10

For additional details, see:
- Chapter 6, "Moving from the OS/VS COBOL run time," on page 61
- Appendix D, "Applications with COBOL and assembler," on page 271

When moving from the VS COBOL II run time

Table 11 on page 29 shows estimated complexity ratings for conversions of specific program attributes.

Table 11. Complexity ratings for programs running under the VS COBOL II run time

Program attribute	Complexity rating
OS/VS COBOL programs compiled with NORES running under VS COBOL II	0
VS COBOL II programs compiled with NORES running under VS COBOL II	0
Loading IGZERRE from an assembler driver	0
Use of IGZEOPT object module (for non-CICS applications)	1
ILC between VS COBOL II programs and PL/I programs, if you link-edit the programs using the PL/I migration tool.	1
CICS online	2
Relies on behavior of RTEREUS runtime option	2
Use of IGZEOPT object module (for CICS applications)	2
Called by assembler routine with a LOAD and Branch	2
ISPF program	3
Use of IGZETUN object module	3
Compiled with NORES and specifies MIXRES runtime option	4
IMS online	4
ILC between VS COBOL II programs and C/370 programs	4
ILC between VS COBOL II programs and PL/I programs	4
ILC with FORTRAN programs	4
Link-editing programs using IGZERRE with the assembler driver	4
Use of ILBOSTP0 with the assembler driver	4
Specifies MIXRES runtime option for OS/VS COBOL programs without using IGZBRDGE	6
Use of BLDL user exit	8
Assembler programs that do not follow normal save area conventions For details, see "Determining requirements for calling and called assembler programs" on page 271.	8

Table 11. Complexity ratings for programs running under the VS COBOL II run time (continued)

Program attribute	Complexity rating
ILC between OS/VS COBOL programs and PL/I programs	8
Assembler program LINKs to a COBOL program when running under a reusable environment	9
Assembler programs that issue a STAE or SPIE	10

For additional details, see:
- Chapter 7, "Moving from the VS COBOL II run time," on page 73
- Appendix D, "Applications with COBOL and assembler," on page 271

Setting up conversion/no-conversion categories

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

There might be some programs that you do not want to move at all, such as:
- Load modules that require the use of the PL/I, C, or FORTRAN portion of Language Environment.
- OS/VS COBOL programs that use ILC with FORTRAN or PL/I

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

Deciding how to phase Language Environment into production mode

When you are ready to use Language Environment in production mode, you need to:
- Determine how to handle multilanguage conversion
- Determine how applications will have access to the library

Multilanguage conversion

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

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

After Language Environment has been installed in LNKLST, all of your COBOL applications will run under Language Environment by default.

Determining how applications will have access to the library

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

LNKLST/LPALST

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

Do not make more than one COBOL runtime library available to your applications at execution 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. When you add Language Environment to LNKLST/LPALST, remove any other COBOL runtime libraries, such as COBLIB, VSCLLIB, and COB2LIB.

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

- Install Language Environment in LNKLST/LPALST on a test or development machine first.
- Use the SETPROG MVS system command to temporarily modify the LNKLST or LPA, without having to IPL the system. For information on using the SETPROG command, see *z/OS MVS System Commands*, SA22-7627.
- IPL over a weekend and install Language Environment in LNKLST/LPALST. Verify over the weekend that your applications run under Language Environment.

Note: Although many elements of z/OS depend on the Language Environment runtime library, z/OS does not require Language Environment to be installed in LNKLST. (However, Language Environment must be installed in the same zone as z/OS.) If you choose not to place Language Environment in LNKLST, you must STEPLIB Language Environment in the individual z/OS PROCs that required Language Environment. For information on which elements require Language Environment, see: *z/OS Program Directory for z/OS Version 1 Release 1*.

STEPLIB

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

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

Note: If you have multiple processors linked together with channel-to-channel connections, you must treat the entire system as one processor and should migrate subsystem by subsystem. In addition to revising your JCL to STEPLIB to the Language Environment run time during initial setup, you might also need to specify CEEDUMP DD if the default allocation for CEEDUMP does not meet your shop's needs. (CEEDUMP is the ddname where Language Environment writes its dump output.) For default CEEDUMP destinations, see "Language Environment formatted dumps" on page 87.

Problems with STEPLIB and IMS programs

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

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

You can use either of the following methods to prevent this problem:
- Install Language Environment into the LNKLST/LPALST.
- Do not preload any runtime routines. (This will slow performance.)

How to minimize the impact:

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

How to recover: If you do notice several different applications abending in the same region, stop the region and restart with these IMS commands:
1. Determine the region number by issuing: `'/DISPLAY ACTIVE'`
2. Stop the region by issuing: `'/STOP REGION region#'`
3. Restart the region by issuing: `'/START REGION region-name'`

STEPLIB example

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

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

     Note for NORES applications: This section does not apply to NORES applications that have not been changed. However, if you change your NORES applications (for example, by link-editing them with Language Environment), they might behave differently than before the link-edit.
   - Run tests with captured data on the Language Environment run time, using the STEPLIB environment, and compare the results to your current run time. Run parallel tests throughout the certification cycle to ensure that your applications produce the same results when run with Language Environment as they did with your current run time.
   - Finally, compile your test applications using Enterprise COBOL. STEPLIB to the Language Environment runtime library, and rerun the certification tests.

2. Install Language Environment on the central development center's system and test.
   - Run parallel tests of the nonconverted versions of your existing applications using STEPLIB to access your current run time.
   - Run all new applications in the Language Environment runtime environment before releasing to production runs.
3. Prepare a backout strategy
   - Save the procedures for installing your current run time in case you need to back out the Language Environment run time.
4. Install the Language Environment run time at one production site.
   - Continue to run parallel tests of the nonconverted versions of your existing applications with your current run time in the STEPLIB environment.
   - Run the Language Environment run time for one month at this production site.
5. Install the Language Environment run time at all production sites.
   - Optional: continue to run parallel tests of the nonconverted versions of your existing applications with your current run time in the STEPLIB environment.
   - Run the Language Environment run time for one month at all production sites.
   - After one month, delete the entire contents of your current runtime library.
6. Compile all new or changed applications with Enterprise COBOL.

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

Setting up a regression testing procedure

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

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

Regression testing will help to identify if there are:

- OS/VS COBOL programs using assembler stubs with SVC LINK in place of COBOL dynamic CALL statements.
- Unclosed files opened by OS/VS COBOL programs or non-COBOL programs, which cause a C03 abend.
- Under CICS, OS/VS COBOL programs using dynamic CALL statements.
- Under CICS, VS COBOL II programs using the CALL statement to call OS/VS COBOL programs.

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

During testing, run your existing applications in parallel on both your current run time and under the Language Environment run time to verify that the results are the same. Take performance measurements of your existing applications to compare with Language Environment.

After the program runs correctly, test it separately and also test it with other programs in a run unit. By testing it against a variety of data, you can exercise all the program processing features to help ensure that there are no unexpected execution differences.

Analyze program output and, if the results are not correct, use Debug Tool or Language Environment dump output to uncover any errors and correct those errors. Make any further changes that you need and then rerun, and, if necessary, continue to debug.

Take performance measurements

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

After you compare runtime performance between Language Environment and your current run time environment and have identified which applications, if any, need performance improvements, you can investigate the methods available to tune your programs and improve performance. For example, you can modify storage values using the Language Environment runtime options. For additional information, see the performance information available on the COBOL Web site. Go to the Library Section at www.ibm.com/software/awdtools/cobol/library.

Cutting over to production use

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

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

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

Chapter 4. Planning to upgrade source programs

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

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

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.

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

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 your product.

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

© Copyright IBM Corp. 1991, 2006

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 or VS COBOL II 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 product number 5648-B05. It is also included in Debug Tool Utilities and Advanced Functions Version 3, Version 4 and Version 5, product number 5655-J18.

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.

Other conversion tools you might want to use include:

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

  The Report Writer Precompiler is product number 5798-DYR. A workstation-based version of the Report Writer Precompiler is also separately orderable, product number 5801-AAR.

These conversion tools are fully described in Appendix C, "Conversion tools for source programs," on page 263.

If you plan to use CCCA, CAMA, 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 the COBOL 68 Standard, COBOL 74 Standard, and COBOL 85 Standard, your programmers will need to be familiar with Language Environment condition handling and Language Environment callable services.

For information on Enterprise COBOL and Language Environment education available through IBM, you can call 1-800-IBM-TEACH. You can also get information directly from Language Environment publications or technical conferences such as 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. By taking an inventory of your applications, you get a detailed picture of the work that is required. You need to take an inventory of:

- Vendor tools, packages, and products
- COBOL applications

~ The Debug Tool Load Module Analyzer can determine the language translator for
~ each object in you load modules. See "Debug Tool Load Module Analyzer" on page 270 for more information.

The Edge Portfolio Analyzer can aid in taking an inventory of your existing load modules, see "The Edge Portfolio Analyzer" on page 270 for more information.

The WebSphere Studio Asset Analyzer for z/OS can aid by analyzing the impact of a code change for an application. See "WebSphere Studio Asset Analyzer" on page 267 for more information.

Taking an inventory of vendor tools, packages, and products

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

- 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.

Taking an inventory of COBOL applications

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

IBM COBOL, VS COBOL II and OS/VS COBOL:

- Programmer responsible
- COBOL Standard level of source program (68, 74, 85)
- Compiler used (ANS COBOL V4, OS/VS COBOL, VS COBOL II, IBM COBOL)
- Compiler options used, especially CMPR2
- Precompiler options used
- Postprocessing options used
- COBOL modules
- COPY library members used in COBOL programs
- Called subprograms
- Calling programs
- Frequency of execution
- Test cases required and available
- Programs containing Report Writer statements

For OS/VS COBOL only:

- Determine which programs use the following features that are not supported by Enterprise COBOL:
  - ISAM files
  - BDAM files
  - Communications feature
- Determine which programs use features that might require the purchase of other products:
  - Report Writer statements require the Report Writer Precompiler
- Determine which programs use features that might have different results under Enterprise COBOL:
  - Variable-length data items (OCCURS DEPENDING ON)
  - Floating-point numeric items
  - Exponentiation
  - Combined abbreviated relation conditions
  - Assembler routines using the high-order bit of Register 13

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

Prioritizing your applications

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

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

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 12 on page 39 are defined as:

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

Based on the complexity ratings shown above (or your own defined complexity ratings), you can now assign a complexity rating to each attribute within a program. Use the highest complexity rating listed as the overall rating for that program. For an application, the highest complexity rating that you assign for any program within the application is the complexity rating for the entire application.

Table 12 shows estimated complexity ratings for conversions of specific program attributes.

Table 12. 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[3]		
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
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[4]	10	M	C
	BDAM	10		C[5]
	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		

Table 12. Complexity ratings for program attribute conversions (continued)

Program attribute	Description of attribute	Complexity rating
CICS		4

Notes:

1. For additional information, see Appendix H, "Preventing file status 39 for QSAM files," on page 305.
2. Non-IBM vendors can recreate COBOL source code from object code.
3. You can use the CICS Application Migration Aid to help convert CICS macro-level programs to command-level programs.
4. Support for ISAM was removed with z/OS 1.7.
5. 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 13 shows one method of relating program complexity ratings to conversion priorities. (The highest priority is "1" and the lowest priority is "6".)

Table 13. Assigning program conversion priorities

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

Table 13. Assigning program conversion priorities (continued)

Conversion priority	Complexity rating	Other considerations
6	9 to 10	Small importance to your organization Very high conversion effort

Consider the following when deciding on conversion priorities:

- If your application is at the limits of the storage available below the 16-MB line, it is a prime candidate for conversion to Enterprise COBOL. With z/OS architecture you can obtain virtual storage constraint relief.
- If the program cannot run under Language Environment, you must convert it. For example, an OS/VS COBOL program that calls or is called by a PL/I program must be upgraded.

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.

Setting up upgrade/no upgrade categories

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

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

- Programs for which you have no source code, that will never need recompilation, and that run correctly under Language Environment.
- Programs of low importance to your organization that run correctly under Language Environment and that would take a very high conversion effort.
- Programs that are being phased out of production.

Note, however, that there might be restrictions on running existing modules mixed with upgraded programs. See Chapter 18, “Adding Enterprise COBOL programs to existing COBOL applications,” on page 219.

Setting up a conversion procedure

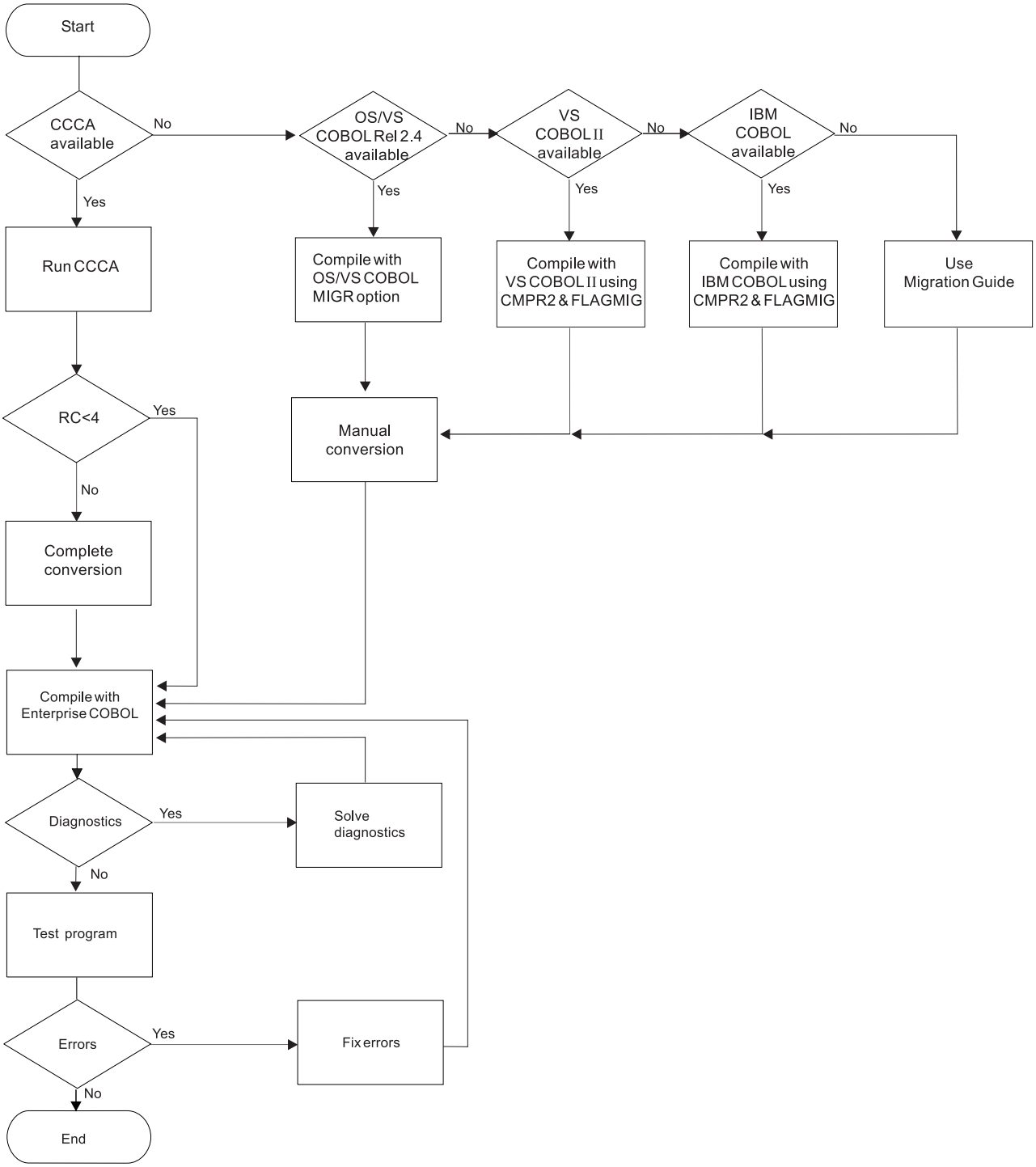
The summaries and diagrams on the following pages outline the steps required to upgrade five types of programs:

- 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.

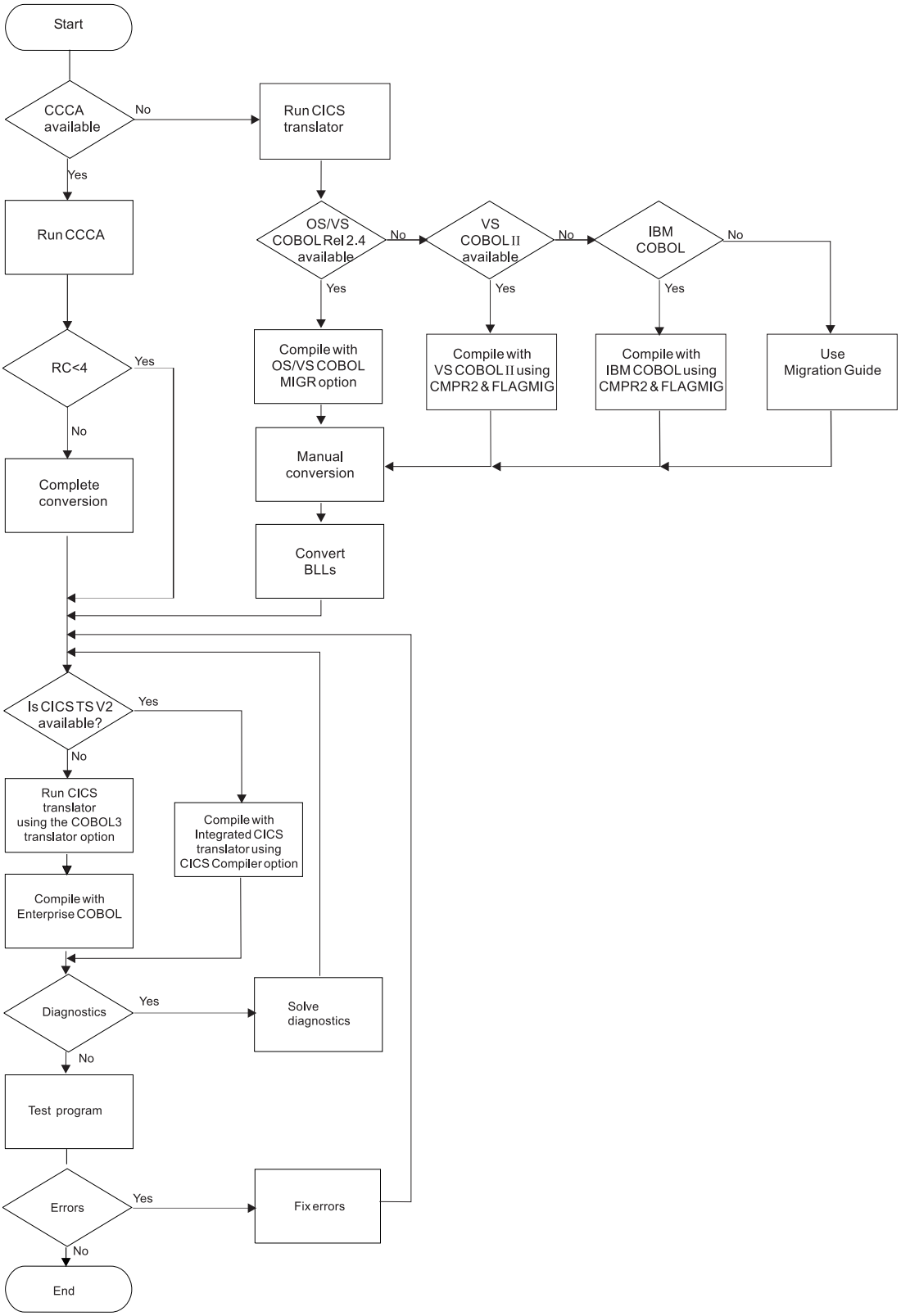
Programs without CICS or Report Writer

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



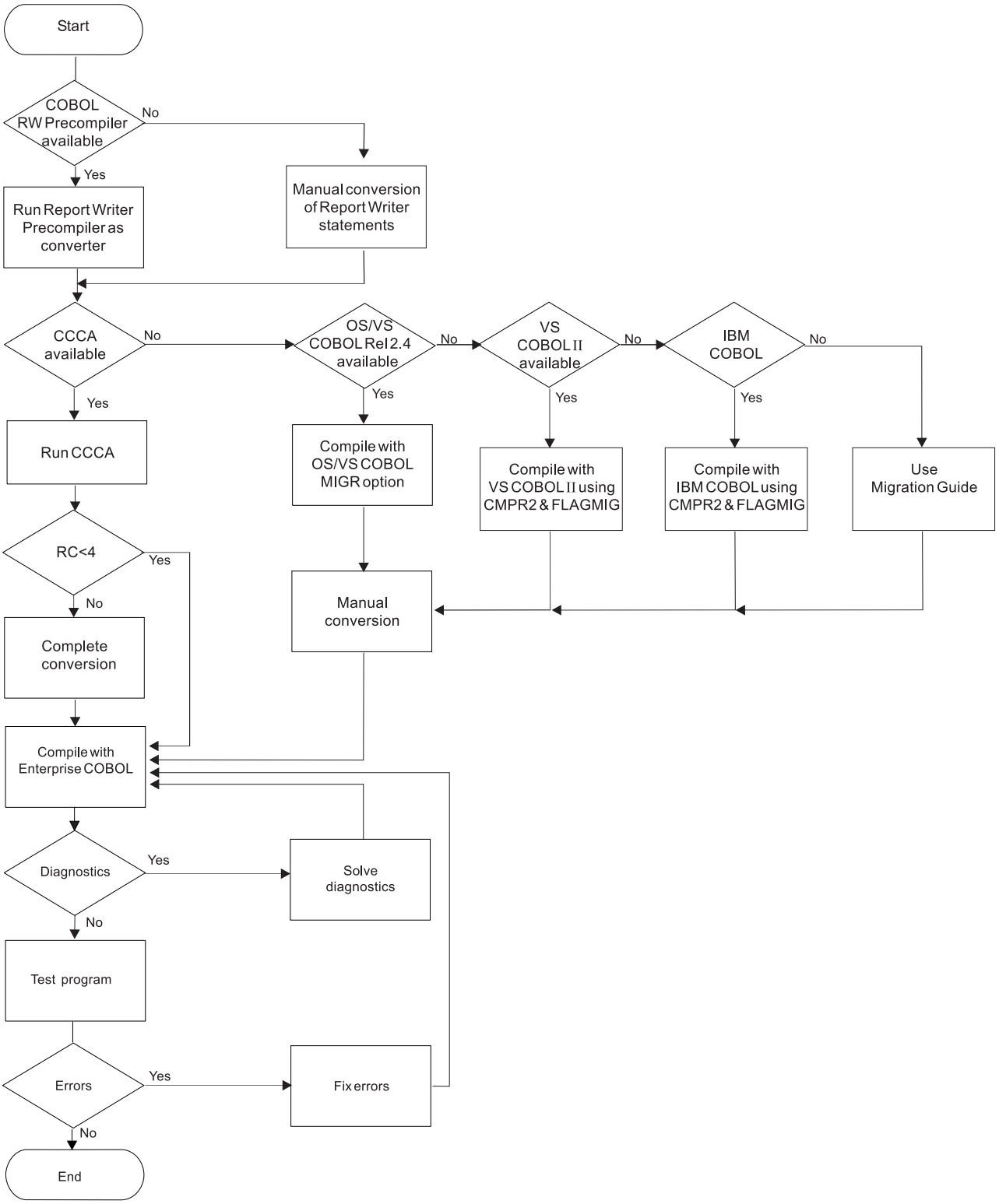
Programs with CICS

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


Start
CCCA available
No
Yes
Run CICS translator
Run CCCA
OS/VS COBOL Rel 2.4 available
No
VS COBOL II available
No
IBM COBOL
No
Yes
Yes
Yes
RC<4
Yes
No
Compile with OS/VS COBOL MIGR option
Compile with VS COBOL II using CMPR2 & FLAGMIG
Compile with IBM COBOL using CMPR2 & FLAGMIG
Use Migration Guide
Complete conversion
Manual conversion
Convert BLLs
Is CICS TS V2 available?
Yes
No
Run CICS translator using the COBOL3 translator option
Compile with Integrated CICS translator using CICS Compiler option
Compile with Enterprise COBOL
Diagnostics
Yes
Solve diagnostics
No
Test program
Errors
Yes
Fix errors
No
End

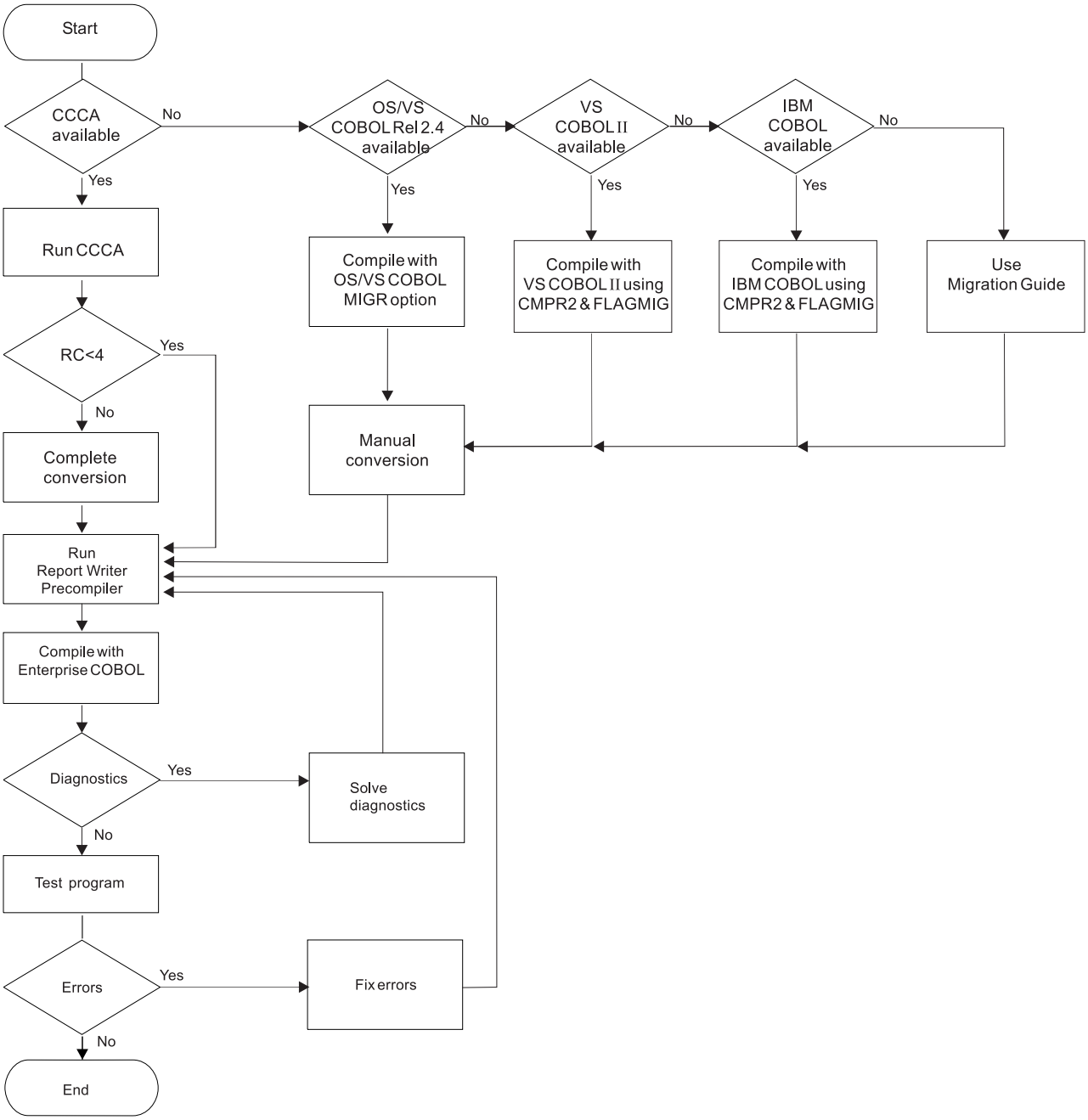
Programs with Report Writer statements to be discarded

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



Programs with Report Writer statements to be retained

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



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 263. 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. (You need the Language Environment run time installed.)

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 H, "Preventing file status 39 for QSAM files," on page 305.
   - Dependency on WORKING-STORAGE being initialized to binary zeros. If you have WORKING-STORAGE zero dependency, specify the Language Environment STORAGE(00) runtime option.
   - 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.

   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 your production system is already using the Language Environment run time. If not, STEPLIB to the Language Environment run time. See "Deciding how to phase Language Environment into production mode" on page 30.)

   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. Moving existing applications to Language Environment

© Copyright IBM Corp. 1991, 2006

Chapter 5. Running existing applications under Language Environment

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

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

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

- Chapter 6, "Moving from the OS/VS COBOL run time," on page 61
- Chapter 7, "Moving from the VS COBOL II run time," on page 73

Set recommended default Language Environment runtime options

The Language Environment IBM-supplied default runtime option settings might not provide the same runtime behavior as the OS/VS COBOL or VS COBOL II runtime options. Because there are differences, this section has two purposes. First, to inform you of the recommended runtime option settings for COBOL programs, so that you can determine which runtime options settings you need to change. Second, to ensure that you do not inadvertently change any default settings that are highly recommended for COBOL programs.

Recommended runtime options for non-CICS applications

Table 14 describes the Language Environment runtime options that are highly recommended for existing non-CICS COBOL applications. For a complete list of Language Environment runtime options, see *Language Environment Programming Reference*.

Table 14. Recommended Language Environment runtime options for non-CICS COBOL applications

Option	LanEnv default setting	Recommended COBOL setting	Comments
ABTERMENC	ABEND	ABEND	ABTERMENC(ABEND) ensures that your application ends with an abend after an abend, program check or severe error occurs, similar to the way OS/VS COBOL and VS COBOL II would end after a problem. For additional VS COBOL II runtime messages considerations, see "Abend codes" on page 86. To obtain a system dump, see "Obtaining a system dump or a CICS transaction dump" on page 58.
CBLOPTS	ON	ON	CBLOPTS(ON) allows the existing COBOL format of the invocation character string to continue working (user parameters followed by the runtime options). This option affects only applications with COBOL as the main program.

© Copyright IBM Corp. 1991, 2006

Table 14. Recommended Language Environment runtime options for non-CICS COBOL applications (continued)

Option	LanEnv default setting	Recommended COBOL setting	Comments
CBLQDA	OFF	OFF	CBLQDA(OFF) suppresses QSAM dynamic allocation for files not available when OPEN OUTPUT, OPEN I-O (optional file), or OPEN EXTEND (optional file) statements are directed to a QSAM file. CBLQDA(OFF) behavior is compatible with programs compiled with OS/VS COBOL, VS COBOL II, Release 2, and VS COBOL II, Release 3, and later compiled with CMPR2, as well as any VS COBOL II NOCMPR2 programs running under VS COBOL II run time with the ZAP from APAR II04562 applied to IGZEQOC. CBLQDA(ON) conforms to the COBOL 85 Standard.
RTEREUS	OFF	OFF	RTEREUS is not recommended as an installation default. If you do use RTEREUS, use it for specific applications only and make sure that you understand the possible side effects and restrictions, for example: • RTEREUS(ON) is ignored if CEEPIPI or DB2 stored procedures are used. • Under Language Environment, RTEREUS(ON) is only supported in a single enclave environment unless you modify the behavior using the IGZERREO CSECT. With the IBM-supplied default setting for COBOL's reusable environment, applications that attempt to create nested enclaves will terminate with error message IGZ0168S. Nested enclaves can be created by applications that use SVC LINK or CMSCALL to invoke applications programs. One example is when an SVC LINK is used to invoke an application program under ISPF that is using ISPF services (such as CALL 'ISPLINK' and ISPF SELECT). • If a Language Environment reusable environment is established (using RTEREUS), attempts to run a C or PL/I main program under Language Environment will fail. For example, when running on ISPF with RTEREUS(ON): – The first program invoked by ISPF is a COBOL program. A Language Environment reusable environment is established. – At some other point, ISPF invokes a PL/I or C program. The initialization of the PL/I or C program will fail. • If a large number of COBOL programs are run under the same MVS task, you can get out of region abends. This is because all storage acquired by Language Environment to run COBOL programs is kept in storage until the MVS task ends or the Language Environment environment is terminated. • Language Environment termination will not be driven unless a STOP RUN is executed to end the enclave. As a result: – Language Environment storage and runtime options reports are not produced by Language Environment. – COBOL files that were not closed by the COBOL program will not be closed by Language Environment, which can result in unpredictable data in the file (for example, records not being written, or the last record being written twice).
ANYHEAP BELOWHEAP HEAP LIBSTACK STACK			These runtime options help you manage storage. For STACK, if any program in your application is running AMODE 24, specify the suboption BELOW (and also specify ALL31(OFF)). If all of the programs in your application are AMODE 31, specify the suboption ABOVE (and also specify ALL31(ON)). On COBOL, the recommended setting for STACK is 64K, 64K, BELOW, KEEP when you use ALL31(OFF) and 64K, 64K, ANY, KEEP when you use ALL31(ON).
TERMTHDACT	TRACE	UADUMP or UATRACE	Use TERMTHDACT(UADUMP), TERMTHADACT(UATRACE), or TERMTHDACT(UAONLY) to receive a system dump under Language Environment when the environment is terminating due to a severe error (for example, a program check or abend). Alternatively, use an abnormal termination exit. See, "Obtaining a system dump or a CICS transaction dump" on page 58 for details.

Table 14. Recommended Language Environment runtime options for non-CICS COBOL applications (continued)

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

Other runtime options affecting non-CICS applications

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

ALL31

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

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

ALL31(ON) allows EXTERNAL data to be allocated anywhere within the 31-bit addressing range and improves runtime performance. The IBM-supplied default for ALL31 is OFF in the OS/390 Language Environment, Version 2 Release 10, and ON in the z/OS Language Environment, Version 1 Release 2, or later.

MSGFILE

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

- SYSABEND
- SYSCOUNT
- SYSDBOUT
- SYSDTERM
- SYSIN
- SYSLIB
- SYSLIN
- SYSLOUT
- SYSPUNCH
- SYSUDUMP

The IBM-supplied default is MSGFILE(SYSOUT).

STORAGE

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

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

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

The IBM-supplied default is STORAGE(NONE,NONE,NONE,8K) in Language Environment, Version 2 Release 10, and is STORAGE(NONE,NONE,NONE,0K) in z/OS Language Environment, Version 1 Release 2 or later.

Recommended runtime options for CICS applications

Table 15 describes the Language Environment runtime options that are highly recommended for existing COBOL CICS applications. On CICS, Language Environment has different default settings than under non-CICS (most of the Language Environment default settings are the same as the recommended COBOL settings).

For a complete list of Language Environment runtime options, see the *Language Environment Programming Reference*.

Table 15. Recommended Language Environment runtime options for COBOL CICS applications

Option	LanEnv default setting on CICS	Recommended COBOL setting	Comments
ABTERMENC	ABEND	ABEND	Ensures you receive abend codes similar to those issued by VS COBOL II or OS/VS COBOL when an abend, program check, or severe error occurs. ABTERMENC(ABEND) ensures that you get a system abend code; to get a system dump, see "Obtaining a system dump or a CICS transaction dump" on page 58.
ALL31	ON	ON	ALL31(ON) allows Language Environment to allocate its control blocks above the line. With ALL31(OFF), Language Environment increases its use of below the 16-MB line storage. For more information on the differences in Language Environment storage usage on CICS for ALL31(OFF) and ALL31(ON), see "Virtual storage requirements" on page 23. ALL31(ON) is the recommended setting for COBOL applications. You can use ALL31(ON) if all of your VS COBOL II, IBM COBOL and Enterprise COBOL programs are AMODE 31, even if you are also running OS/VS COBOL programs in your CICS regions. (Load modules containing only OS/VS COBOL and assembler programs are AMODE 24 and are not affected by the setting of ALL31.) **Note:** To run with ALL31(ON), every program in the Language Environment enclave must be AMODE 31, including assembler programs, unless the main program is AMODE 24. In this case, Language Environment will automatically switch to ALL31(OFF) for the run unit (enclave). Use ALL31(OFF) if your load modules contain VS COBOL II, IBM COBOL, or Enterprise COBOL programs and assembler programs that require AMODE 24 and are in run units where the first program in the run unit was AMODE 31.
ANYHEAP BELOWHEAP HEAP LIBSTACK STACK	See *Language Environment Installation and Customization*.	Default settings	Language Environment provides these runtime options to help manage storage.

Table 15. Recommended Language Environment runtime options for COBOL CICS applications (continued)

Option	LanEnv default setting on CICS	Recommended COBOL setting	Comments
TERMTHDACT	TRACE	UADUMP or UATRACE	Use TERMTHDACT(UADUMP), TERMTHDACT(UATRACE), or TERMTHDACT(UAONLY) to receive a transaction dump under Language Environment when the environment is terminating due to a severe error (for example, a program check or abend). Alternatively, use an abnormal termination exit. See "Obtaining a system dump or a CICS transaction dump" on page 58 for details. You might not want to use TERMTHDACT(DUMP), TERMTHDACT(TRACE), TERMTHDACT(UADUMP), or TERMTHDACT(UATRACE) in production, because these TERMTHDACT suboptions can cause a lot of time to be spent writing Language Environment dump data to transient queue data CESE when a transaction abends. If a traceback CEEDUMP is not needed by the application environment use TERMTHDACT(MSG) to eliminate the performance overhead of writing formatted CEEDUMPs to the CESE CICS transient data queue.
TRAP	ON	ON	TRAP specifies how Language Environment routines handle abends and program interrupts. In order for applications to run successfully, you must specify TRAP(ON).

Other runtime options affecting CICS applications

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

CBLPSHPOP

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

CBLPSHPOP(ON) ensures you receive behavior that is compatible with the behavior of running VS COBOL II programs with the VS COBOL II run time. CBLPSHPOP(OFF) can give performance benefits. For details, see "CICS HANDLE commands and the CBLPSHPOP runtime option" on page 98.

STORAGE

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

To receive the Language Environment-equivalent of the VS COBOL II WSCLEAR runtime option, set the first suboption of the Language Environment STORAGE runtime option to 00. For example, STORAGE(00,NONE,NONE,0K).

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

Invoking existing applications

To access Language Environment, you will need to change the procedures that you use for invoking applications. The procedures required for non-CICS applications are different than the procedures for CICS applications.

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

For non-CICS applications

The following sections detail the changes required for non-CICS applications. For considerations for programs run on IMS, see Appendix L, "IMS considerations," on page 319. For more information on how to prepare and run your programs with Language Environment, see the *Language Environment Programming Guide*.

Specify the correct library

To invoke existing applications when running under Language Environment, you need to replace your current library with the Language Environment SCEERUN library.

Specify alternate DDNAMES (optional)

With Language Environment, you can indicate the destination for Language Environment output by changing the ddname in the MSGFILE runtime option to the ddname you want. Table 16 lists the default ddnames for Language Environment output.

Table 16. Specification of new DDNAMEs

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

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

- On z/OS: SYSOUT=*
- On TSO: ALLOC DD(SYSOUT) DA(*)

Remove DDNAMES no longer required (optional)

The following ddnames are not required when running with Language Environment:

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

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

For CICS applications

To run Language Environment on CICS, you need to perform several required steps. For details on how to invoke COBOL applications running on CICS under Language Environment, including how to specify the Language Environment runtime library SCEERUN, see *Language Environment for z/OS Customization*

Output differences when using Language Environment on CICS

Under CICS, Language Environment output goes to a transient data queue named CESE. Each record written to the file has a header that includes the terminal ID, the transaction ID, date, and time. Table 17 lists the types of Language Environment output and location.

Table 17. Location of Language Environment output under CICS

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

Link-editing existing applications

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

Include the Language Environment SCEELKED in the SYSLIB concatenation.

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

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

There are a couple of ways to avoid these problems:

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

the conflict with the Language Environment subroutine is preferable to placing you private subroutine libraries ahead of SCEELKED.

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

- "Determining which programs require link-editing" on page 62, if running under the OS/VS COBOL run time
- "Determining which programs require link-editing" on page 74, if running under the VS COBOL II run time

Obtaining a system dump or a CICS transaction dump

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

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

Method 1: Specify the TERMTHDACT runtime option

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

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

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

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

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

Method 2: Specify an abnormal termination exit

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

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

Abnormal termination exit under non-CICS
For non-CICS applications, the sample abnormal termination exit is SAMPDAT1, as shown in Figure 3. SAMPDAT1 provides system abend dumps.

Abnormal termination exit under CICS
For CICS applications, the sample abnormal termination exit is SAMPDAT2, as shown in Figure 4 on page 60. SAMPDAT2 provides transaction dumps.

For information on using an abnormal termination exit, see *Language Environment for z/OS Customization*

Abnormal termination exit (non-CICS)

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

Figure 3. Non-CICS abnormal termination exit sample

Abnormal termination exit (CICS)

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

Figure 4. CICS abnormal termination exit sample

Getting compatible abend behavior

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

ABTERMENC(ABEND) ensures you receive system abend codes similar to those issued by VS COBOL II or OS/VS COBOL when an abend, program check, or severe error occurs. (To get a system dump, see "Obtaining a system dump or a CICS transaction dump" on page 58.)

To get user abend codes similar to what you received under VS COBOL II, like U1xxx (where xxx is the IGZ message number), for unhandled Language Environment software generated conditions, do *one* of the following:

- Modify the assembler user exit (CEEBXITA) by copying the code from the COBOL sample assembler user exit (CEEBX05A) into CEEBXITA.
- Use the sample user condition handler CEEWUCHA as described in "Using CEEWUCHA" on page 86.

Ensuring the compatibility of return-code values

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

- If you specify the ABTERMENC(RETCODE) runtime option
- If you modify the Language Environment assembler user exit to manipulate the return-code value

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

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

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

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

- Appendix D, "Applications with COBOL and assembler," on page 271
- Appendix L, "IMS considerations," on page 319

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

RES An application comprised of programs compiled with RES.

NORES

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

OR

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

- A VS COBOL II program
- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- An Enterprise COBOL program
- The IGZCBSN or IGZCBSO bootstrap routine

CICS An application that runs under CICS.

NORES linked

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

- A VS COBOL II program
- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- An Enterprise COBOL program

© Copyright IBM Corp. 1991, 2006

- The IGZCBSN or IGZCBSO bootstrap routine

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

Determining which programs require link-editing

This section applies to the following compiler options:
- RES
- NORES

To determine which programs you need to link-edit with Language Environment, you need to know:
1. If the program was compiled with the RES or NORES compiler option
2. If the program uses a reusable environment (established by ILBOSTP0)

For a summary table of load modules with OS/VS COBOL programs that require link-edit, see Table 45 on page 223.

For additional information on relink-editing a COBOL load module with Language Environment, see Appendix J, "Link-edit example," on page 311.

Applications with COBOL programs compiled with RES

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

Applications with COBOL programs compiled with NORES

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

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

~ **ISAM:** Support was dropped by DFSMS with z/OS 1.7.

If you do link-edit programs compiled with NORES with Language Environment, see Chapter 8, "Link-editing applications with Language Environment," on page 103 for details on possible changes in behavior.

Applications with COBOL programs compiled with RES and NORES

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

- A COBOL/370 program
- A COBOL for MVS & VM program
- A COBOL for OS/390 & VM program
- An Enterprise COBOL program
- The IGZCBSN bootstrap routine
- The IGZCBSO bootstrap routine

Determining which programs require upgrading

This section applies to the following compiler options:
- RES
- NORES
- CICS

Note: We recommend upgrading all OS/VS COBOL programs to Enterprise COBOL whenever upgrading is discussed in this chapter.

On CICS

OS/VS COBOL programs that issue dynamic CALL statements when running under CICS will abend with abend code U3504. You can either upgrade all the COBOL programs in the run unit to Enterprise COBOL or recode the programs without dynamic CALL statements.

You must upgrade OS/VS COBOL programs if:

- You plan to use CICS Transaction Server, Version 2 Release 2 or later, and you want to translate the OS/VS COBOL programs with the CICS translator.

On non-CICS

You must upgrade OS/VS COBOL programs if:

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

You might also want to upgrade the following:

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

ILC with PL/I

OS/VS COBOL programs that call or are called by PL/I must be upgraded.

ILC with FORTRAN

OS/VS COBOL programs that call or are called by FORTRAN must be upgraded.

OS/VS COBOL programs in more than one enclave

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

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

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

Comparing runtime options and specification methods

This section applies to the following compiler options:
- RES
- NORES
- NORES Linked

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

Specifying Language Environment runtime options

Language Environment provides various methods for specifying runtime options. Three mechanisms are available to specify options:
- runtime option CSECTs
  - CEEDOPT for installation-wide defaults
  - CEEROPT for region-wide defaults
  - CEEUOPT for application-specific defaults
- Invocation procedures
- Assembler user exit

To determine which of the Language Environment methods you can use for existing applications, you need to know whether the main program is compiled with RES or NORES and if the main program has been link-edited with Language Environment. Table 18 lists the specification methods available to specify and override runtime options under Language Environment.

Table 18. Methods available for specifying runtime options for OS/VS COBOL programs

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

Note:

1. In this case, the NORES programs exhibit RES behavior after link-edit. For more information about the impact of link-editing with Language Environment, see Chapter 8, "Link-editing applications with Language Environment," on page 103.

Installation-wide defaults

The CEEDOPT (non-CICS) and CEECOPT (CICS) assembler files contain default values for all Language Environment runtime options. At installation time, you can

edit this file and select defaults that will apply to all applications running in the common environment (except OS/VS COBOL programs under CICS).

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

For information on how to set installation-wide default runtime options for Language Environment, see *Language Environment for z/OS Customization*

Region-wide defaults

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

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

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

CEEUOPT application-specific defaults

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

Invocation procedures

For OS/VS COBOL programs that are called directly from the operating system, you can specify runtime options at program invocation using the appropriate operating system mechanism. On z/OS you can specify runtime options in the PARM parameter of the JCL EXEC statement.

Note: On CICS, you cannot use the PARM parameters to specify runtime options.

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

Order of precedence

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

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

Comparing OS/VS COBOL and Language Environment runtime options

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

Table 19. Comparison of OS/VS COBOL and Language Environment runtime options

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

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

This section applies to the following compiler options:
- RES
- NORES Linked

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

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

Other environments

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

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

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

Running in a reusable runtime environment

This section applies to the following compiler options:
- RES
- NORES Linked

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

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

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

  Note: Modifying the behavior using the IGZERREO CSECT does not remove the restriction that OS/VS COBOL cannot be in more than one enclave.
- After the reusable environment has been established, the assembler driver can call only a COBOL program or non-Language Environment-conforming assembler program. COBOL can then call any supported language as follows:
  - A COBOL program that is called by the assembler driver cannot use a static CALL statement to call another language. A dynamic CALL to another language is supported.
  - Any COBOL program that is dynamically called by another COBOL program can use a static CALL statement or a dynamic CALL statement to another language.
- The first high-level language program must be a COBOL program (assembler is not considered a high-level language).
- If an assembler driver is statically linked with, and calls, ILBOSTP0 to provide the reusable environment, it must be link-edited with Language Environment.

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

You can improve the performance when using the COBOL reusable environment or you can allow nested enclaves when running in a reusable environment by modifying the IGZERREO CSECT. For additional information, see the *Language Environment for z/OS Customization*

Using ILBOSTP0

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

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

Managing dump services

This section applies to the following compiler options:
- RES
- CICS
- NORES Linked

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

OS/VS COBOL symbolic dumps

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

System storage dumps and CICS transaction dumps

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

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

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

To obtain a system storage dump or CICS transaction dump under Language Environment, see “Obtaining a system dump or a CICS transaction dump” on page 58.

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

Language Environment formatted dumps

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

The Language Environment formatted dump can contain various information, including information on:
- Symbolic dump of variables, arguments, and registers

- File control blocks
- File buffers
- Runtime control blocks
- Storage used by the program

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

Dump destination on non-CICS

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

If the CEEDUMP file is needed and is not defined, Language Environment dynamically allocates one for you, with the default being SYSOUT=*.

Dump destination on CICS

All runtime output from Language Environment is directed to a CICS transient data queue named CESE. Each record written to the file has a header that includes the terminal ID, the transaction ID, and the date and time.

Using ILBOABN0 to force an abend

This section applies to the following compiler options:
- RES
- NORES Linked

With OS/VS COBOL, you can use a CALL to ILBOABN0 to force an immediate abend and obtain a system dump. With Language Environment, a CALL to ILBOABN0 will continue to force an immediate abend. To produce a system dump, see “Obtaining a system dump or a CICS transaction dump” on page 58.

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

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

Using SORT or MERGE in OS/VS COBOL programs

This section applies to the following compiler options:
- RES
- NORES Linked

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

- Language Environment will not produce a Language Environment dump if a program check or abnormal termination occurs while in a SORT or MERGE user exit.
- OS/VS COBOL debug data will not be produced if a program check or abnormal termination occurs while in a SORT or MERGE user exit.
- Language Environment will not perform environment cleanup if a program check or abnormal termination occurs while in a SORT or MERGE user exit.

- While in an input or output procedure, if an assembler program is called, the assembler program cannot call a COBOL program.
- If a program check occurs while in a SORT or MERGE user exit, the application will end with abend U4036. For information on how to find the state of the registers and PSW at the time of the program check, see *Language Environment Debugging Guide and Run-Time Messages*.

Understanding SYSOUT output changes

This section applies to the following compiler options:
- RES
- NORES Linked

With Language Environment, SYSOUT output is different than with OS/VS COBOL. Differences are with:
- SYSOUT output sent to a file with RECFM=FB or to any other destination when DCB=(RECFM=FB) is specified on the DD
- OS/VS COBOL trace output sequence

SYSOUT output with RECFM=FB

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

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

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

OS/VS COBOL trace output sequence

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

Figure 5 shows the differences between OS/VS COBOL and Language Environment.

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

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

Figure 5. TRACE output under Language Environment compared to OS/VS COBOL

Communicating with other languages

This section applies to the following compiler options:
- RES
- NORES
- CICS
- NORES Linked

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

Interlanguage communication is defined as programs that call or are called by other high-level languages. Assembler is not considered a high-level language; thus, calls to and from assembler programs are not considered ILC. For details on Language Environment's support for calls involving COBOL programs and Assembler programs, see:
- "Runtime support for assembler COBOL calls on non-CICS" on page 273
- "Runtime support for assembler COBOL calls on CICS" on page 274

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

Table 20 shows the action required for applications with existing ILC:

Table 20. Action required for existing applications using ILC

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

Additional CICS considerations

OS/VS COBOL programs running with Language Environment under CICS run in a subset of the full Language Environment runtime environment. When you run an OS/VS COBOL application on CICS, the environment that is established for a run unit by Language Environment supports only OS/VS COBOL.

Chapter 7. Moving from the VS COBOL II run time

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

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

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

- Appendix D, “Applications with COBOL and assembler,” on page 271
- Appendix L, “IMS considerations,” on page 319

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

RES An application comprised of programs compiled with RES.

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

OR

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

- A VS COBOL II program
- An IBM COBOL program
- An Enterprise COBOL program
- The IGZCBSN or IGZCBSO bootstrap routine
- A program using the IGZBRDGE routine

CICS An application that runs under CICS.

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

- A VS COBOL II program

© Copyright IBM Corp. 1991, 2006

- An IBM COBOL program
- An Enterprise COBOL program
- The IGZCBSN or IGZCBSO bootstrap routine
- A program using the IGZBRDGE routine

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

Determining which programs require link-editing

This section applies to the following compiler options:
- RES
- NORES

To determine which programs you need to link-edit with Language Environment, you need to know:
- If the program was compiled with the RES or NORES compiler option
- If the program specifies the MIXRES runtime option
- If the program uses a reusable environment (established by ILBOSTP0)
- If the program uses ILC
- If you statically call IGZCA2D or IGZCD2A

For additional information on load modules with VS COBOL II programs that must be link-edited, see "VS COBOL II considerations" on page 224.

For additional information on link-editing a COBOL load module with Language Environment, see Appendix J, "Link-edit example," on page 311.

Applications with COBOL programs compiled with RES

If you call ILBOSTP0 to establish a reusable environment (that is, you establish an assembler language program as the main program by link-editing it with, and calling, ILBOSTP0), you must link-edit the assembler program with Language Environment or you can change your assembler program to use CEEPIPI.

Applications with COBOL programs compiled with NORES

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

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

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

   If the program was not link-edited with an object module produced using the IGZBRDGE macro, and if this is the only load module in an application or the first load module in an application with multiple load modules, after link-editing with Language Environment, it will exhibit NORES behavior. (That is, the load module will not have access to Language Environment services.)

The reason you must link-edit this application with Language Environment is to ensure that the OS/VS COBOL initiated termination, like STOP RUN, will work.

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

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

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

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

Programs that use ILC

Existing VS COBOL II load modules that use ILC with PL/I, C, or FORTRAN require link-editing. For details, see:
- "COBOL and PL/I" on page 93
- "COBOL and C/370" on page 93
- "COBOL and FORTRAN" on page 92

Statically calling IGZCA2D or IGZCD2A

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

Determining which programs require upgrading

This section applies to the following compiler options:
- RES
- NORES
- CICS

Note: We recommend upgrading all OS/VS COBOL and VS COBOL II programs to Enterprise COBOL whenever upgrading is discussed in this chapter.

CICS

Under CICS, if you have any VS COBOL II programs that use a CALL statement to call OS/VS COBOL programs, you must upgrade the OS/VS COBOL program.

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

Non-CICS

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

You must upgrade OS/VS COBOL programs if:

- OS/VS COBOL programs use ILC with PL/I
- OS/VS COBOL programs use ILC with FORTRAN
- OS/VS COBOL programs are contained in more than one enclave (or run unit)

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

ILC with PL/I

OS/VS COBOL programs that call or are called by PL/I must be upgraded.

ILC with FORTRAN

OS/VS COBOL RES programs that call or are called by FORTRAN must be upgraded.

OS/VS COBOL programs in more than one enclave

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

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

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

Comparing runtime options and specification methods

This section applies to the following compiler options:

- RES
- NORES
- CICS
- NORES Linked

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

Note: If you compile any of your VS COBOL II programs with Enterprise COBOL at the same time that you are moving to Language Environment, then read Chapter 18, "Adding Enterprise COBOL programs to existing COBOL applications," on page 219.

Specifying Language Environment runtime options

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

- runtime option CSECTs:

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

In addition, the VS COBOL II application-specific runtime options module (IGZEOPT) is available to load modules comprised of VS COBOL II programs compiled with RES. IGZEOPT is ignored under CICS.

To determine which of the Language Environment methods you can use for existing applications, you need to know whether the main program is compiled with RES or NORES and if the main program has been link-edited with Language Environment. Table 21 lists the specification methods available to specify and override runtime options under Language Environment.

Table 21. Methods available for specifying runtime options

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

Note:

1. IGZEOPT is ignored when running on CICS.
2. In this case, the NORES programs will exhibit "RES behavior" after link-editing. For more information on the implication of link-editing with Language Environment, see Chapter 8, "Link-editing applications with Language Environment," on page 103.
3. See Table 22 on page 79 for details on when the runtime options specified using IGZEOPT are in effect.
4. VS COBOL II NORES programs that do not specify the MIXRES runtime option and that are not link-edited with Language Environment can continue to use IGZEOPT.

Installation-wide defaults

The CEEDOPT (non-CICS) and CEECOPT (CICS) assembler files contain default values for all Language Environment runtime options. At installation time, you can edit this file and select defaults that will apply to all applications running in the common environment.

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

For information on how to set installation-wide default runtime options see the *Language Environment for z/OS Customization*

Region-wide defaults

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

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

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

Application-specific defaults

Language Environment provides an assembler file, CEEUOPT, that you can edit to select runtime options that will apply to a specific application or program.

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

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

Invocation procedures

runtime options can also be specified at program invocation using the appropriate operating system mechanism. (COBOL programs must be called directly from the operating system to specify runtime options at invocation.) On z/OS, you can specify runtime options in the PARM parameter of the JCL EXEC statement.

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

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

Order of precedence

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

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

Specifying VS COBOL II runtime options

The following information is included only as a compatibility aid to assist you in running existing applications without having to upgrade to Enterprise COBOL or

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

Applications with COBOL programs compiled with RES (non-CICS)

The VS COBOL II application-specific runtime options module, IGZEOPT, is available to RES applications, running under Language Environment. After you link-edit an application with Language Environment, it is possible for both IGZEOPT and CEEUOPT to exist in the same application. Table 22 shows the relationship between IGZEOPT and CEEUOPT for applications that are link-edited with Language Environment.

Table 22. Application-specific options for existing applications compiled with VS COBOL II and link-edited with Language Environment

IGZEOPT present	CEEUOPT present	Comments
No	No	The runtime options are not affected.
Yes	No	The runtime options specified in IGZEOPT are mapped to the appropriate Language Environment runtime option.
No	Yes	The runtime options specified in CEEUOPT are in affect.
Yes	Yes	The runtime options specified in CEEUOPT are in affect. IGZEOPT is ignored. No error message is issued.

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

Applications with COBOL programs compiled with NORES (non-CICS)

The default VS COBOL II runtime options module, IGZEOPD, is available to NORES applications that have been link-edited with the VS COBOL II runtime library. IGZEOPT is also available to VS COBOL II NORES applications, if the MIXRES runtime option is not used. After these applications have been link-edited with Language Environment, IGZEOPD and IGZEOPT are ignored.

Applications running on CICS

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

Comparing VS COBOL II and Language Environment options

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

Table 23. Comparison of VS COBOL II and Language Environment runtime options

VS COBOL II option	Language Environment option	Comment
AIXBLD	AIXBLD	Invokes the access method services (AMS) for VSAM indexed and relative data sets to complete the file and index definition procedures for COBOL routines. With Language Environment, you do not need to specify ddname SYSPRINT when running on z/OS. The access method services (AMS) messages are directed to the ddname specified in the Language Environment MSGFILE runtime option (default is SYSOUT). AIXBLD is not applicable on CICS.
DEBUG	DEBUG	This option is specified with the same syntax and supported in the same manner under Language Environment as in VS COBOL II.
LANGUAGE	NATLANG	NATLANG replaces LANGUAGE, which is a VS COBOL II installation option. Language Environment provides the ability to select a national language at run time, as well as at installation time, using the NATLANG option.
LIBKEEP		LIBKEEP is not supported. The Language Environment Library Routine Retention facility can provide similar function as LIBKEEP. For more information, see "Existing applications using LIBKEEP" on page 94. The LIBKEEP option is not applicable on CICS.
MIXRES		MIXRES is not supported under Language Environment. The MIXRES option is not applicable on CICS.
RTEREUS	RTEREUS	RTEREUS is supported in the same manner under Language Environment as in VS COBOL II. RTEREUS is not recommended as an installation default under Language Environment. For important considerations before using RTEREUS, see "Other runtime options affecting non-CICS applications" on page 53, and "Precautions if establishing a reusable environment under IMS" on page 82. The RTEREUS option is not applicable on CICS.
SIMVRD	SIMVRD	This option is specified with the same syntax and supported in the same manner in Language Environment as in VS COBOL II. The SIMVRD option is not applicable on CICS.
SPOUT	RPTOPTS(ON) RPTSTG(ON)	SPOUT is mapped to the Language Environment options RPTOPTS and RPTSTG. Both storage reports (RPTSTG) and options report (RPTOPTS) are directed to the ddname specified in MSGFILE (default SYSOUT). The report formats are different under Language Environment than with VS COBOL II. For more information, see "Determining storage tuning changes" on page 95.
SSRANGE	CHECK(ON)	SSRANGE is mapped directly to CHECK.
STAE	TRAP(ON)	STAE is mapped directly to TRAP(ON). For non-CICS applications, TRAP(ON) causes both ESTAE and ESPIE to be issued. Under VS COBOL II, STAE only causes ESTAE to be issued. Note, that under VS COBOL II STAE is optional. Under Language Environment, Release 1.9 (OS/390 V2R6) or later, in non-CICS, you can specify TRAP(ON,NOSPIE) in which case only an ESTAE will be issued.
UPSI	UPSI	This option is specified with the same syntax and supported in the same manner in Language Environment as in VS COBOL II.

Table 23. Comparison of VS COBOL II and Language Environment runtime options (continued)

VS COBOL II option	Language Environment option	Comment
WSCLEAR	STORAGE	WSCLEAR is not supported. For similar function, use STORAGE(00,NONE,NONE,8K) for non-CICS applications and STORAGE(00,NONE,NONE,0K) for CICS applications.

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

This section applies to the following compiler options:
- RES
- NORES Linked

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

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

Other environments

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

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

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

Running in a reusable runtime environment

This section applies to the following compiler options:
- RES
- NORES Linked

Language Environment continues to provide compatibility support for pre-Language Environment methods for managing a reusable runtime environment. Three of these methods include:

- Calling IGZERRE from an assembler program.
- Calling ILBOSTP0 from an assembler program.
- Specifying the RTEREUS runtime option.

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

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

  Note: Modifying the behavior using the IGZERREO CSECT does not remove the restriction that OS/VS COBOL cannot be in more than one enclave.
- After the reusable environment has been established, the assembler driver can call only a COBOL program or non-Language Environment conforming assembler program. COBOL can then call any supported language as follows:
  - A COBOL program that is called by the assembler driver cannot use a static CALL statement to call another language. A dynamic CALL to another language is supported.
  - Any COBOL program that is dynamically called by another COBOL program can use a static CALL statement or a dynamic CALL statement to another language.
- The first high-level language program must be a COBOL program (assembler is not considered a high-level language).
- If an assembler driver is statically linked with, and calls, IGZERRE or ILBOSTP0 to provide the reusable environment, it must be link-edited with Language Environment.
- If a COBOL NORES program is running with the RTEREUS runtime option in effect, it must be link-edited with Language Environment.

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

You can improve the performance when using the COBOL reusable environment or you can allow nested enclaves when running in a reusable environment by modifying the IGZERREO CSECT. For additional information, see the *Language Environment for z/OS Customization*.

Precautions if establishing a reusable environment under IMS

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

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

For additional details on using RTEREUS, see “Recommended runtime options for non-CICS applications” on page 51.

Using IGZERRE

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

On return from IGZERRE, register 15 contains a return code. Table 24 shows the changes when running with Language Environment:

Table 24. Return code changes for IGZERRE

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

Using ILBOSTP0

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

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

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

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

Using RTEREUS

The RTEREUS runtime option for initializing the COBOL reusable environment is supported by Language Environment, although it is not recommended as an installation default. For important considerations, see the RTEREUS discussion under “Recommended runtime options for non-CICS applications” on page 51. For RES environments, the behavior of RTEREUS is the same as it was in VS COBOL II with the exception of using SVC LINK (for example, ISPF ″SELECT PGM()″).

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

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

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

For important IMS considerations see "Precautions if establishing a reusable environment under IMS" on page 82.

Managing messages, abend codes, and dump services

This section applies to the following compiler options:
- RES
- CICS
- NORES Linked

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

Runtime messages

Two factors affect runtime messages for applications that run with Language Environment:

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

Messages prefixed with IKF

There are no differences for OS/VS COBOL messages. The message prefix, number, severity, and content remain unchanged. Also, the destination remains the same, even when the program is link-edited with Language Environment. The messages appear synchronously and are written using OS write-to-programmer. This is true regardless of the state of the runtime initialization process.

Messages prefixed with IGZ

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

- On non-CICS, if the runtime initialization process is not complete, messages are routed as directed by OS write-to-programmer.
- On non-CICS, if the runtime initialization process is complete, then messages are directed to the file *ddname* specified on the Language Environment MSGFILE runtime option, which has a default of SYSOUT. On z/OS, SYSOUT=* is the default.

  If MSGFILE ddname is not defined, it will be dynamically allocated with Language Environment defined file attributes. For details, see *Language Environment Programming Guide*.
- On CICS, all runtime output from Language Environment is directed to a CICS transient data queue named CESE. Each record written to the file has a header that includes the terminal ID, the transaction ID, date and time. Runtime output is no longer written to a temporary storage queue.

Note: To direct the runtime output to a temporary storage queue, you can define the transient data queue CESE as an intrapartition data queue and specify a transaction that reads records from CESE, strips off the header information, and writes the rest of the record to a temporary storage queue.

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

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

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

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

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

Timing of abend for runtime detected errors

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

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

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

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

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

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

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

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

When you use CEEWUCHA under Language Environment, the following events occur when there is a runtime detected error (with ABTERMENC(ABEND) in effect):

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

Abend codes

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

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

Using CEEWUCHA

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

CEEWUCHA contains code to do the following:

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

To use CEEWUCHA:

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

Dump services

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

OS/VS COBOL symbolic dumps

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

VS COBOL II FDUMPs

For VS COBOL II programs compiled with the FDUMP option, you could get a formatted dump of COBOL WORKING-STORAGE. VS COBOL II FDUMPs were directed to the SYSDBOUT data set. For details on dump destination under Language Environment, see "Language Environment formatted dumps" on page 87.

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

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

Language Environment formatted dumps

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

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

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

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

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

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

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

Dump destination on CICS: All runtime output from Language Environment is directed to a CICS transient data queue named CESE. Each record written to the file has a header that includes the terminal ID, the transaction ID, and the date and time.

System storage dumps

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

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

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

To obtain a system storage dump, see "Obtaining a system dump or a CICS transaction dump" on page 58.

On z/OS, dump output is directed to SYSUDUMP, SYSMDUMP, or SYSABEND. On other systems, VS COBOL II runtime dump output is directed to SYSABOUT. (SYSABOUT is not used under Language Environment, even for VS COBOL II programs.)

Using ILBOABN0 to force an abend

This section applies to the following compiler options:
- RES
- NORES Linked

Under OS/VS COBOL and VS COBOL II, you can use a CALL to ILBOABN0 to force an immediate abend and obtain a system dump. Under Language Environment, a CALL to ILBOABN0 will continue to force an immediate abend. To produce a system dump, see "Obtaining a system dump or a CICS transaction dump" on page 58.

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

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

Using SORT or MERGE

This section applies to the following compiler options:
- RES
- NORES Linked

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

In OS/VS COBOL programs

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

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

In VS COBOL II subprograms

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

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

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

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

For example:

```
⋮
    SORT SD01
       ASCENDING KEY A3
       USING INP1
       OUTPUT PROCEDURE OUTPRO1.
⋮
OUTPRO1 SECTION.
```

```
⋮
    GOBACK.
    * With Language Environment, this GOBACK statement will
    * cause condition IGZ0012S.
```

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

Understanding SYSOUT output changes

This section applies to the following compiler options:
- RES
- NORES Linked

Under Language Environment, DISPLAY UPON SYSOUT is different than in OS/VS COBOL and VS COBOL II as follows:
- SYSOUT output sent to a file with RECFM=FB or to any other destination when DCB=(RECFM=FB) is specified on the DD
- OS/VS COBOL trace output sequence

DISPLAY UPON SYSOUT and DD definitions

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

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

SYSOUT output with RECFM=FB

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

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

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

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

OS/VS COBOL trace output sequence

The trace output sequence has changed under Language Environment. In Language Environment, only *one* trace entry per record appears, instead of

numerous trace entries per record as in OS/VS COBOL programs running under VS COBOL II. This change is especially helpful with intervening DISPLAY SYSOUT output and error messages.

Figure 6 shows the difference between the OS/VS COBOL and Language Environment formats.

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

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

Language Environment :

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

Figure 6. TRACE output under Language Environment compared to OS/VS COBOL

Communicating with other languages

This section applies to the following compiler options:
- RES
- NORES
- CICS
- NORES Linked

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

Interlanguage communication is defined as programs that call or are called by other high-level languages. Assembler is not considered a high-level language; thus, calls to and from assembler programs are not considered ILC. For details on Language Environment's support for calls involving COBOL programs and assembler programs, see:
- "Runtime support for assembler COBOL calls on non-CICS" on page 273
- "Runtime support for assembler COBOL calls on CICS" on page 274

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

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

General ILC considerations

COBOL DISPLAY output (non-CICS)

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

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

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

Effect of STOP RUN (non-CICS)

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

COBOL and FORTRAN

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

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

- VS COBOL II, Release 3 and later (static CALL statements only)
- IBM COBOL
- Enterprise COBOL
- VS FORTRAN, Version 1 (except for modules compiled with Release 1 or Release 2 and either are subprograms that receive character arguments, or pass character arguments to subprograms).
- VS FORTRAN, Version 2 (except modules compiled with Release 5 or Release 6) that:
  Contain parallel language constructs
  Specify the PARALLEL compiler option
  Invoke parallel callable services (PEORIG, PEPOST, PEWAIT, PETERM, PLCOND, PLFREE, PLLOCK, PLORIG, or PLTERM)
- FORTRAN IV G1
- FORTRAN IV H Extended

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

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

COBOL and PL/I

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

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

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

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

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

Difference in behavior for dynamically called RENT programs

VS COBOL II programs use a different copy of WORKING-STORAGE for each call under the VS COBOL II run time, however, under Language Environment the same copy of WORKING-STORAGE is used for each call. These different situations occur when the following conditions are true:

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

In addition, the programs enter their initial state when run under the VS COBOL II runtime library. However, when running under Language Environment the programs are entered in their last-used state, unless there is an intervening CANCEL.

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

COBOL and C/370

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

- VS COBOL II, Version 1 Release 3, and later
- IBM COBOL
- Enterprise COBOL
- C/370, Version 1 (5688-040)
- C/370, Version 2 (5688-187)
- OS/390 C/C++
- z/OS C/C++

You must link-edit any existing, supported C and COBOL ILC applications either with:
- Language Environment
- The C conversion tool (APAR PN74931) while running on C/370, Version 2. (For the C conversion tool to work on Language Environment, Release 5, you must also install APAR PN77483. The function is in the base for Language Environment Release 6 and later.)

You can use either method, except in the following cases, when you must link-edit with Language Environment:
- When the application contains COBOL NORES programs.
- When the application requires COBOL program-specific runtime options or space tuning.

Difference in behavior for dynamically called RENT programs

VS COBOL II programs use a different copy of WORKING-STORAGE for each call under the VS COBOL II run time, however, under Language Environment the same copy of WORKING-STORAGE is used for each call. These different situations occur when the following conditions are true:
- Compiled with the RENT option
- Dynamically called from OS/VS COBOL, VS COBOL II, IBM COBOL, or Enterprise COBOL programs
- Fetched and called by C

In addition, the programs enter their initial state when run under the VS COBOL II runtime library. However, when running under Language Environment the programs are entered in their last-used state, unless there is an intervening CANCEL.

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

Initializing the runtime environment

This section applies to the following compiler options:
- RES
- NORES Linked

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

Existing applications using LIBKEEP

The VS COBOL II LIBKEEP runtime option was used to enhance runtime performance by maintaining the partition level of the runtime environment between calls to COBOL main programs. The Language Environment runtime environment does not support the LIBKEEP runtime option.

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

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

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

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

Considerations for Language Environment preinitialization

Language Environment preinitialization services for the main program environment allow you to initialize the runtime environment for a main program by using the CEEPIPI(INIT_MAIN ...) service, to call programs as main by using the CEEPIPI(CALL ...) service, and to terminate the preinitialized environment using the CEEPIPI(TERM ...) service.

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

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

When running with the Language Environment preinitialization facility for the main program environment, those files plus MSGFILE and dump files will not be closed until CEEPIPI(TERM) is issued to terminate the preinitialized environment.

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

Determining storage tuning changes

This section applies to the following compiler options:

- RES
- CICS
- NORES Linked

The existing methods (the IGZTUNE macro and the SPOUT and WSCLEAR runtime options) for space management and tuning are not available under Language Environment. Language Environment services now control space

management and tuning using the RPTOPTS, RPTSTG, STORAGE, ANYHEAP, BELOWHEAP, HEAP, LIBSTACK, and STACK runtime options.

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

Alternatives to IGZTUNE

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

Note: For applications using IGZETUN, see "Using CEEWUCHA" on page 86 for details on how to suppress the warning-level message and the overhead of writing it to the Language Environment MSGFILE.

This is accomplished by using the following five storage management runtime options:

HEAP Manages heap storage for user data such as WORKING-STORAGE, EXTERNAL data, and EXTERNAL file information.

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

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

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

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

The RPTSTG runtime option provides the optimum values to use when specifying the storage management options. (You will need to first use the RPTSTG option to generate a report of storage used in your program or run unit. You can then use this report to determine the values that you need to specify in the five Language Environment storage options in order to achieve the space tuning purpose.)

Do not use the values from IGZTUNE when specifying the Language Environment tuning options. These values might not be optimal for tuning storage usage within the Language Environment environment. For recommended Language Environment storage option settings, see "Set recommended default Language Environment runtime options" on page 51.

Considerations for SPOUT output

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

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

Additional CICS considerations

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

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

Performance considerations

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

- Set the CICS SIT option RUWAPOOL to YES. Doing so can reduce the CICS GETMAIN and FREEMAIN activity when CICS LINK is used.
- Tune your Language Environment storage runtime options. Doing so can keep the CICS GETMAIN and FREEMAIN activity to a minimum, and avoids the overhead that occurs when an increment is needed. If you are using CICS Transaction Server, Version 1 Release 3 or later, the automatic storage tuning is performed when you set the CICS SIT option AUTODST to YES.
- You might not want to use TERMTHDACT(DUMP) and TERMTHDACT(UADUMP) in production, because these TERMTHDACT suboptions can cause a lot of time to be spent writing Language Environment dump data to transient data queue CESE when a transaction abends.
- Do not run with runtime options RPTOPTS(ON) or RPTSTG(ON) in production. These options have a significant impact on performance.
- If you are getting any warning messages written by Language Environment to CESE, make changes so that they are not issued.

SORT interface change

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

WORKING-STORAGE limits

The WORKING-STORAGE limits for VS COBOL II program running on Language Environment are different than when running on the VS COBOL II run time as shown in Table 25 on page 98. This difference affects applications that are

developed using Language Environment, and run in production using VS COBOL II.

Table 25. WORKING-STORAGE limits for VS COBOL II programs on CICS

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

VS COBOL II NORENT programs

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

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

IGZETUN or IGZEOPT and MSGFILE

When you run on Language Environment and have either IGZETUN or IGZEOPT link-edited with your main program, Language Environment writes warning message IGZ0014W to the Language Environment MSGFILE each time a main program that contains IGZETUN or IGZEOPT is run. Writing the IGZ0014W message to the MSGFILE can cause a significant amount of additional system resource to be used. To suppress the writing of IGZ0014W, see "Using CEEWUCHA" on page 86.

CICS HANDLE commands and the CBLPSHPOP runtime option

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

The CICS PUSH HANDLE and POP HANDLE commands suspend/restore the current effects of the following CICS commands:

- IGNORE CONDITION
- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION

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

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

CBLPSHPOP(OFF) prevents the issuing of the PUSH HANDLE and POP HANDLE commands and must be used with care to avoid compatibility problems when upgrading. If you do not issue a CICS PUSH HANDLE command before a call to a COBOL subroutine, the subroutine will inherit any IGNORE CONDITION or HANDLE command from the caller. If the subroutine then issues an IGNORE CONDITION or HANDLE command, the caller will inherit their effects upon return.

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

- The program that issues the CALL statement contains either of these CICS commands used for condition handling in CICS:
  - CICS IGNORE CONDITION
  - CICS HANDLE ABEND PROGRAM(program)
- The COBOL subroutine contains any of the following CICS commands:
  - CICS IGNORE CONDITION
  - CICS HANDLE ABEND
  - CICS HANDLE AID
  - CICS HANDLE CONDITION
  - CICS PUSH HANDLE
  - CICS POP HANDLE

Note that CICS HANDLE...(label) commands in the caller will not cause compatibility problems with CBLPSHPOP(OFF) as long as your program does not contain any of the statements listed above.

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

Example 1—no effect on compatibility

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PGM1
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
    EXEC CICS HANDLE ABEND LABEL(LBL1) END-EXEC.
    CALL "PGM2" USING DFHEIBLK DFHCOMMAREA
    EXEC CICS RETURN END-EXEC.
LBL1.
    EXEC CICS RETURN END-EXEC.


IDENTIFICATION DIVISION.
PROGRAM-ID. PGM2
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DATA1    PIC 9 VALUE 0.
```

```
 01  DATA2    PIC 9 VALUE 1.
 PROCEDURE DIVISION.
* Force a DIVIDE-BY-ZERO exception
     COMPUTE DATA1 = DATA2 / DATA1
     GOBACK.
```

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

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

Example 2—effect on compatibility

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PGM1
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  DATA1    PIC 9 VALUE 0.
 01  DATA2    PIC 9 VALUE 1.
 PROCEDURE DIVISION.
     EXEC CICS HANDLE ABEND LABEL(LBL1) END-EXEC.
     CALL "PGM2" USING DFHEIBLK DFHCOMMAREA
* Force a DIVIDE-BY-ZERO exception
     COMPUTE DATA1 = DATA2 / DATA1
     EXEC CICS RETURN END-EXEC.
 LBL1.
     EXEC CICS RETURN END-EXEC.

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

In this example the CBLPSHPOP option will affect compatibility with VS COBOL II. With CBLPSHPOP(ON), PGM1's HANDLE ABEND label is saved and restored across the call to PGM2, and the divide by zero exception is handled with a branch to LBL1. The program terminates normally.

With CBLPSHPOP(OFF), PGM2's HANDLE ABEND command is in effect on the return to PGM1. The exception caused by the divide by zero in PGM1 causes CICS to ask Language Environment to branch to LBL1A in PGM2. Language Environment prevents this branch, and the program terminates with an ASRA abend.

DISPLAY statement

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

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

CLER transaction

The CICS transaction CLER allows you to display all the current Language Environment runtime options for a region, and to also have the capability to modify a subset of these options.

Undocumented VS COBOL II extensions

An input or output procedure that is referenced on a COBOL SORT or MERGE statement cannot directly or indirectly invoke another SORT or MERGE. When you use the VS COBOL II runtime library, a nested SORT or a MERGE statement like this was not diagnosed. However, Language Environment, Version 2 Release 7 (with PQ39908) or later detects the nested SORT or MERGE in a VS COBOL II program and issues message IGZ0173S before terminating the program.

Chapter 8. Link-editing applications with Language Environment

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

- Programs compiled with NORES
- Programs compiled with RES

For additional information on relink-editing a COBOL load module with Language Environment, see Appendix J, "Link-edit example," on page 311.

Applications comprised of NORES programs

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

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

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

However, in some cases, after you link-edit an OS/VS COBOL NORES application with Language Environment, the application will exhibit RES behavior. Table 26 lists what can cause this change in behavior.

Table 26. Program attributes causing changes in application behavior

Attribute	Comments
Contains a VS COBOL II program	Applications with a VS COBOL II program will exhibit RES behavior after being link-edited with Language Environment.
Contains an IBM COBOL or Enterprise COBOL program	Any applications with an Enterprise COBOL, COBOL for OS/390 & VM, COBOL for MVS & VM, or COBOL/370 program will exhibit RES behavior after being link-edited with Language Environment. For additional information, see Chapter 18, "Adding Enterprise COBOL programs to existing COBOL applications," on page 219.
Contains the IGZCBSN bootstrap routine	IGZCBSN is the COBOL/370 bootstrap routine.
Contains the IGZCBSO bootstrap routine	IGZCBSO is the COBOL for MVS & VM, COBOL for OS/390 & VM, and Enterprise COBOL bootstrap routine.
Contains a program using an object module produced using the IGZBRDGE macro.	An object module produced by using the IGZBRDGE macro is normally used to convert static CALL statements to dynamic CALL statements. However, just the presence of an object module produced by using IGZBRDGE causes the behavior to change, regardless of how it is being used.

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

© Copyright IBM Corp. 1991, 2006

Implications of becoming RES-like

Only one of the attributes listed in Table 26 on page 103 needs to be present to cause NORES applications to act as RES applications. If your NORES application does become RES-like, you have additional considerations:

- Many of the considerations for programs that have been compiled with RES now apply to the NORES applications that you have link-edited with Language Environment (as indicated with the "NORES Linked" category of the **This section applies to:** box at the beginning of each section in Chapter 6, "Moving from the OS/VS COBOL run time" and Chapter 7, "Moving from the VS COBOL II run time").
- Additional Language Environment services are available. For example, if you want to get a report of storage used by your program:
  - Before link-editing the NORES application with Language Environment, Language Environment was not initialized. You could not generate a storage report.
  - After link-editing the NORES application with Language Environment, Language Environment is initialized. You can now request a storage report by specifying the RPTSTG runtime option.
- Performance will be impacted.

Applications comprised of RES programs

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

VS COBOL II RES programs that are link-edited with Language Environment are enabled to use the Language Environment application-specific runtime options CSECT, CEEUOPT. Because IGZEOPT and CEEUOPT can exist in the same application, see Table 22 on page 79 for details on which method will be in effect.

Chapter 9. Upgrading Language Environment release levels

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

- Change in behavior for DATA(31) programs under OS/390, Version 2 Release 9, or later
- Missing CEEDUMP for applications with assembler programs that use the DUMP macro under OS/390, Version 2 Release 8
- Change in file handling for COBOL programs with RECORDING MODE U under OS/390, Version 2 Release 10
- Calling between assembler and COBOL programs with different AMODEs under OS/390, Version 2 Release 9 or later
- Referencing symbolic feedback tokens

CSECT replace: You do not need to relink modules to replace CEE* CSECTS or IGZ* CSECTS (by using REPLACE linkage-editor control statements) when you move from one release of Language Environment to another release of Language Environment. Doing so is not recommended.

Change in behavior for DATA(31) programs under OS/390, Version 2 Release 9, or later

Although users requested WORKING-STORAGE and parameter lists above the 16-MB line with the DATA(31) compiler option, there were cases in which storage was acquired from below the 16-MB line under Language Environment for OS/390, Version 2 Release 8 or earlier. If these programs dynamically call AMODE 24 programs, then moving from Language Environment for OS/390, Version 2 Release 8 or earlier to Language Environment for OS/390, Version 2 Release 9 or later will now result in runtime error message IGZ0033S.

Under Language Environment for OS/390, Version 2 Release 8, or earlier, WORKING-STORAGE for COBOL programs compiled with DATA(31) was acquired from a HEAP segment that was allocated from BELOW storage. Under Language Environment for OS/390, Version 2 Release 9 or later, WORKING-STORAGE for COBOL programs compiled with DATA(31) is acquired from a HEAP segment that is allocated from ANYWHERE storage (which can be above or below the 16-MB line). Consider the following examples where this change will impact an application:

- A previous HEAP request in an enclave must specifically request HEAP storage below the 16-MB line. For example, a DATA(24) COBOL program that requested this type of HEAP storage caused the Language Environment heap manager to allocate a HEAP segment from BELOW storage.
- A subsequent DATA(31) COBOL program must request HEAP storage that can be satisfied in a HEAP segment allocated from BELOW storage. In order for this to occur, the WORKING-STORAGE can not be very large.

When a DATA(31) COBOL program runs under Language Environment for OS/390, Version 2 Release 8 or earlier, the HEAP is acquired from BELOW storage. When the same program runs under Language Environment for OS/390, Version 2 Release 9 or later, the HEAP is acquired from ANYWHERE storage.

© Copyright IBM Corp. 1991, 2006

If a call is made to an AMODE 24 program, the following considerations apply under Language Environment for OS/390, Version 2 Release 9 or later:

- If the call is a COBOL dynamic call to an AMODE 24 program, runtime error message IGZ0033S will occur.
- If the call to the AMODE 24 program is called using any method other than a COBOL dynamic call, an addressing exception can occur when the program attempts to access the data passed.

Missing CEEDUMP for applications with assembler programs that use the DUMP macro under OS/390, Version 2 Release 8

The following section applies when you are moving from:

- Language Environment for OS/390, Version 2 Release 7 or earlier to Language Environment for OS/390, Version 2 Release 8 or later with the PTFs for APAR PQ38656 applied, or
- Language Environment for OS/390, Version 2 Release 7 or earlier to Language Environment for z/OS

Assembler programs that use the ABEND macro will not trigger CEEDUMP unless they use the DUMP parameter (ABEND, DUMP). Previous Language Environment releases would get a CEEDUMP with or without the DUMP parameter. NODUMP is the default for the DUMP macro. This change was introduced to Language Environment for OS/390, Version 2 Releases 8, 9, and 10 with APAR PQ38656.

Change in file handling for COBOL programs with RECORDING MODE U under OS/390, Version 2 Release 10

When you move from Language Environment for OS/390, Version 2 Release 9 or earlier to Language Environment for OS/390, Version 2 Release 10, or Language Environment for z/OS, COBOL programs that process RECFM=VB or FB data sets with RECORDING MODE U might work differently. The application program might require some modification to work correctly in either of the following cases:

- The application reads a RECFM=VB or FB data set as RECORDING MODE U and needs to receive an individual record rather than the entire block.
- The application writes to a RECFM=VB data set defined as RECORDING MODE U and needs to write multiple records per block.

This combination of RECORDING MODE and RECFM created a mismatch that coincidentally worked in some cases under Language Environment for OS/390, Version 2 Release 9 or earlier, and some applications exploited this behavior. Programs that are compiled with CMPR2 do not have this problem, but inconsistent behavior might occur if the PTF for PQ49479 is not installed.

To determine if this problem is present in your applications, search for RECORDING MODE U in your source programs. If you do not find references to RECORDING MODE U, the chances that you will run into this problem are minimal. However, if you do have programs that use RECORDING MODE U, you need to view the DD statement and maybe the data set attributes for the file.

- If the DD statement for the file defined as RECORDING MODE U contains RECFM=U, the application will probably not require any modification to run under Language Environment for OS/390, Version 2 Release 10, or Language

Environment for z/OS. The JCL RECFM=U will override the data set label so that the source program and attributes that are used to open the file will be consistent.

- If the DD statement for the file defined as RECORDING MODE U contains RECFM=VB (V or FB also apply), you must modify the application before it will run under Language Environment for OS/390, Version 2 Release 10 or Language Environment for z/OS.
- If the DD statement for the file defined as RECORDING MODE U does not contain any RECFM= specification, you need to view the data set attributes. If the RECFM for the data set is RECFM=VB, V, or FB, you must modify the application before it will run under Language Environment for OS/390, Version 2 Release 10 or Language Environment for z/OS.

Your changes must ensure that the RECORDING MODE of the file is consistent with the RECFM of the data set when the file is opened. Following are your options for modifying your applications:

- You can modify your source to use RECORDING MODE V and format 3 of the RECORD clause "RECORD IS VARYING FROM n TO nn DEPENDING ON data-name-1" in order to process variable-length records. A read done from a file that was defined using this record clause would receive the length of the record that was just read in data-name-1. If the file is variable-blocked, add a BLOCK CONTAINS clause.
- You can continue to process the file as RECORDING MODE U, but you must ensure that the application can handle a true RECFM=U file. To verify that the application handles the file correctly, use a JCL override of RECFM=U and run the application. Examine the results of the processing to ensure that the application did correctly process the records that were read and written with true format U processing.
- Ensure that the format of the data set is RECFM=U. To set this format, you can use a JCL override of the true RECFM or define format U as the true RECFM for the data set. Before using this option, you must verify that your application can correctly process the records that are returned with true format U processing.

In true format U processing, each read from the file receives an entire block (because format U records are written so that each block is a single record) which will be passed back to the application. If the data set is variable blocked (RECFM=VB) or fixed blocked (RECFM=FB), the block that is returned to the application will consist of multiple records that must be deblocked by the application. Additionally each write to a format U file will create a new block for each record. If a write is done to an existing file that was originally written as RECFM=VB or RECFM=FB, the blocks written as RECFM=U will contain a single record, whereas those written as RECFM=FB or RECFM=VB can contain multiple records. The RECFM will also be changed to U if a write is done.

A mismatch between the RECORDING MODE of the file and RECFM of the data set can also exist when RECORDING MODE V or F is coded in your COBOL source and the data set is RECFM=U. Although this coding might currently work, it is recommended that the RECORDING MODE and RECFM are changed to match. In the future, changes to Language Environment might cause your applications with a mismatch between the RECORDING MODE for the file and the RECFM for the data set to fail.

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

When you move to Language Environment for OS/390 & VM, Version 2 Release 10 or later, the amount of space that is required by an output file might differ from previous releases. In cases where the output data set has RECFM=U, but the file was defined to the COBOL program as fixed blocked or variable blocked, and the blocks are defined to contain multiple records, the output file might be much larger than expected.

The mismatch between the RECFM of the data set and the RECORDING MODE and BLOCK CONTAINS clause in the COBOL program causes COBOL to set the blocksize to the length that is defined in the COBOL program, but QSAM processes as it would for RECFM=U, which causes a single record to be written to each block. The result is that the data set can fill up with blocks of a size intended to contain multiple records, but each containing only a single record. This result will cause the file to be larger than expected, and can lead to B37 or D37 out-of-space abends when a program writes to or closes the file if there are no available extents. To fix this problem, make the RECFM of the output data set consistent with the RECORDING MODE and BLOCK CONTAINS clause in the COBOL program. QSAM will then write the records to the output file in a manner consistent with the COBOL file definition.

Calling between assembler and COBOL under OS/390, Version 2 Release 9 or later

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

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

With Language Environment for OS/390, Version 2 Release 8 or earlier, the behavior for CALL statements between COBOL and assembler is as follows:

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

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

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

Dynamic calls to assembler programs under Language Environment for z/OS, Version 1 Release 2 or later

When you migrate from earlier Language Environment releases to Language Environment for z/OS, Version 1 Release 2 or later, you need to be aware of a change to the RMODE of the COBOL dynamic call routine IGZCFCC. The change to the RMODE for IGZCFCC might cause problems if a dynamically called assembler program returns in a different AMODE from the AMODE in which it was entered. IGZCFCC is used only by COBOL programs that were compiled with COBOL for MVS & VM, Release 2 or later. IGZCFCC is not used by programs that were compiled with the VS COBOL II or OS/VS COBOL compilers.

Beginning in z/OS Language Environment, Version 1 Release 2, IGZCFCC was changed from RMODE 24 to RMODE 31 and incorporated into the IGZCPAC load module, which resides above the line. When a dynamic call is made to an AMODE 31 program, the return address is to IGZCFCC.

In Language Environment releases prior to z/OS, Version 1 Release 2, an assembler program that was entered in AMODE 31 could change the AMODE to 24 and still return successfully to IGZCFCC because it was loaded below the line. With IGZCFCC residing above the line in Language Environment for z/OS, Version 1 Release 2 and later, this no longer works. If an assembler program is entered in AMODE 31, switches to AMODE 24 and then does a load and branch to return to its caller, the high-order byte of the 31-bit return address can be truncated, resulting in a bad branch, ABEND0C4, ABEND0C1, and so forth.

You can fix the problem in either of the following ways:

- Compile and link the assembler program as AMODE 24. COBOL will handle the mode switching.
- Modify the assembler program so that the AMODE that was in use when the program was entered is restored when the program returns to COBOL.

Diagnosis of a COBOL program that has an FD entry for a VSAM file on CICS with Language Environment for z/OS, V1R2 or later

Beginning in Language Environment for z/OS V1R2, the environmentally sensitive modules for VSAM (IGZEVIO, IGZEVOP, IGZEVO2) are no longer available as load modules in SCEERUN. This change eliminates non-CICS versions of some environmentally sensitive modules, which can cause error message IGZ0096C to be issued when a COBOL program with an FD entry for a VSAM file is run under CICS.

There have always been restrictions on the use of native COBOL file I-O in a CICS environment (as opposed to CICS file I-O). The use of FILE-CONTROL entries in the ENVIRONMENT DIVISION or a FILE SECTION in the DATA DIVISION is not

allowed in CICS applications, unless these language elements are used for SORT statements. Even in cases where the program logic avoids executing the COBOL I-O statements in a CICS environment, the presence of native COBOL I-O elements is not supported. Previous releases of the Language Environment runtime did not diagnose the presence of FD entries for VSAM files in a CICS environment.

Referencing symbolic feedback tokens

Between Language Environment, Version 1 Release 3 and Version Release 4, changes were made to 11 condition tokens in CEEIGZCT. If your programs reference any of the following changed symbolic feedback tokens in CEEIGZCT you must recompile the programs when upgrading to Language Environment, Release 4 or later:

CEE36U	CEE372	CEE58Q
CEE36V	CEE373	CEE58R
CEE37O	CEE374	CEE58S
CEE371	CEE375	

Part 4. Upgrading source programs

© Copyright IBM Corp. 1991, 2006

Chapter 10. Upgrading OS/VS COBOL source programs

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

Future considerations

In CICS TS, Version 2 Release 2 or later, you can no longer use the CICS Translator for OS/VS COBOL programs. As soon as possible, upgrade to Enterprise COBOL any OS/VS COBOL programs that need to change to run under CICS, Version 2 Release 2 or later.

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

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

- *VS COBOL for OS/VS Reference GC26-3857-04*
- *Enterprise COBOL Language Reference SC27-1408*

Note: *VS COBOL for OS/VS Reference* is no longer available from IBM.

Notes:

1. There are special considerations for new, changed, or unsupported language elements when you are running under CICS. For details, see Chapter 17, "CICS conversion considerations for COBOL source," on page 209.
2. In the following sections, any reference to COBOL 68 Standard is a reference to the COBOL language supported by IBM Full American National Standard COBOL, Version 4 (Program 5734-CB2), or to LANGLVL(1) of OS/VS COBOL (Program 5740-CB1).
3. The information provided in this chapter, and throughout this manual, is intended for OS/VS COBOL, Release 2.4, with the latest service updates applied.

Comparing OS/VS COBOL to 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 27 on page 114 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.

© Copyright IBM Corp. 1991, 2006

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

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

Table 27. Language element differences between OS/VS COBOL and Enterprise COBOL

Language element	Conversion tool	Page
Abbreviated combined relation conditions		128
ACCEPT statement		128
ALPHABETIC class changes	CCCA	136
ALPHABET clause changes—ALPHABET key word	CCCA	136
Area A, periods in	CCCA	132
Arithmetic statement changes		136
ASSIGN . . . OR	CCCA	121
ASSIGN TO *integer system-name*	CCCA	121
ASSIGN . . . FOR MULTIPLE REEL /UNIT	CCCA	121
ASSIGN clause changes—*assignment-name* forms	CCCA	137
B symbol in PICTURE clause—changes in evaluation		137
BDAM file handling	CCCA[1]	120
BLANK WHEN ZERO clause and asterisk (*) override		128
CALL identifier statement—B symbol in PICTURE clause		137
CALL statement changes—procedure names and file names in USING phrase		137
CANCEL statement—B symbol in PICTURE clause		137
CLOSE . . . FOR REMOVAL statement		129
CLOSE statement—WITH POSITIONING, DISP phrases	CCCA	121
Combined abbreviated relation condition changes	CCCA	138
Comparing group to numeric packed-decimal item		129
COPY statement with associated names	CCCA	139
Communication feature		121
CURRENCY-SIGN clause changes—'/', '=', and 'L' characters		140
CURRENT-DATE special register	CCCA	121
DIVIDE . . . ON SIZE ERROR—change in intermediate results		145
Dynamic CALL statements when running on CICS		129
Dynamic CALL statements to programs with alternate entry points without an intervening CANCEL		140
EXAMINE statement	CCCA	122
EXHIBIT statement	CCCA	123
EXIT PROGRAM/GOBACK statement changes		140
FILE STATUS clause changes	CCCA	140
FILE-LIMIT clause of the FILE-CONTROL paragraph	CCCA	124
Flow of control, no terminating statement		129
FOR MULTIPLE REEL /UNIT	CCCA	121

Table 27. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
GIVING phrase of USE AFTER STANDARD ERROR declarative	CCCA	124
IF . . . OTHERWISE statement changes	CCCA	142
Index names—nonunique		130
INSPECT statement—PROGRAM COLLATING SEQUENCE clause		146
IS as an optional word		145
ISAM file handling	CCCA	119
JUSTIFIED clause changes	CCCA	143
LABEL RECORDS clause with TOTALING/TOTALED AREA	CCCA	124
LABEL RECORD IS statement		130
MOVE statement—binary value and DISPLAY value		130
MOVE statements and comparisons—scaling changes		143
MOVE CORRESPONDING statement	CCCA	130
MOVE statement—multiple TO specification		131
MOVE ALL—TO PIC 99		131
MOVE statement—warning message for numeric truncation		131
MULTIPLY ... ON SIZE ERROR—change in intermediate results		145
Nonunique program-ID names	CCCA	133
NOTE statement	CCCA	125
Numeric class test on group items		144
Numeric data changes		144
Numeric-editing changes (PICTURE clause)		133
OCCURS clause (order of phrases)		131
OCCURS DEPENDING ON—ASCENDING and DESCENDING KEY phrases		144
OCCURS DEPENDING ON—value for receiving items changed	CCCA	144
ON statement	CCCA	125
ON SIZE ERROR phrase—changes in intermediate results		145
OPEN statement failing for QSAM files (file status 39)		125
OPEN statement failing for VSAM files (file status 39)		125
OPEN statement with LEAVE, REREAD, and DISP phrases	CCCA	125
OPEN REVERSED statement		132
OTHERWISE clause changes		142
Paragraph names not allowed as parameters		137
PERFORM statement—changes in the VARYING and AFTER phrases		146
PERFORM statement—second UNTIL		132
Periods, consecutive in any division		132
Periods in Area A	CCCA	132
Periods missing on paragraphs	CCCA	133
Periods missing at the end of SD, FD, or RD		133
PICTURE clause (numeric-editing changes)		133
PROGRAM COLLATING SEQUENCE clause changes		146

Table 27. Language element differences between OS/VS COBOL and Enterprise COBOL (continued)

Language element	Conversion tool	Page
Program-ID names, nonunique	CCCA	133
Qualification - using the same phrase repeatedly		133
READ statement - redefined record keys in the KEY phrase		133
READ and RETURN statement changes—INTO phrase		146
READY TRACE and RESET TRACE statements	CCCA	125
RECORD CONTAINS n CHARACTERS clause		133
RECORD KEY phrase and ALTERNATE RECORD KEY phrase		133
REDEFINES clause in SD or FD entries	CCCA	133
REDEFINES clause with tables		134
Relation conditions	CCCA	134
REMARKS paragraph	CCCA	126
RENAMES clause—nonunique, nonqualified data names		134
Report Writer statements	Report Writer Precompiler	118
RERUN clause changes		146
RESERVE clause changes	CCCA	147
Reserved word list changes	CCCA	147
SEARCH statement changes	CCCA	147
Segmentation changes—PERFORM statement in independent segments		148
SELECT statement without a corresponding FD		134
SELECT OPTIONAL clause changes	CCCA	148
SORT special registers		148
SORT verb		134
SORT or MERGE		135
Source language debugging changes		148
START . . . USING KEY statement	CCCA	127
STRING statement—PROGRAM COLLATING SEQUENCE clause		146
STRING statement—sending field identifier		135
Subscripts out of range—flagged at compile-time		149
THEN as a statement connector	CCCA	127
TIME-OF-DAY special register	CCCA	127
TOTALING/TOTALED AREA phrases in LABEL RECORDS clause	CCCA	124
TRANSFORM statement	CCCA	127
UNSTRING statement—PROGRAM COLLATING SEQUENCE clause		146
UNSTRING statement—coding with 'OR', 'IS', or a numeric edited item	CCCA	135
UNSTRING statement—multiple INTO phrases		136
UNSTRING statements—subscript evaluation changes		149
UPSI switches	CCCA	150
USE AFTER STANDARD ERROR—GIVING phrase	CCCA	124
USE BEFORE STANDARD LABEL statement	CCCA	128

Table 27. 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	136
VALUE clause—condition names	CCCA	150
WHEN-COMPILED special register	CCCA	151
WRITE AFTER POSITIONING statement	CCCA	151

Note:

1. This is a partial conversion for handling BDAM files.

Using conversion tools to convert programs to COBOL 85 Standard

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

- The COBOL conversion tool (CCCA)
- The OS/VS COBOL MIGR compiler option
- This *Migration Guide*

A brief description of these conversion tools follows. See Appendix C, "Conversion tools for source programs," on page 263 for additional information.

Note: Non-IBM tools are also available to help automate the conversion to the COBOL 85 Standard. For details, see "Vendor products" on page 270.

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 267 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 263 lists the items flagged by MIGR. A complete description of MIGR-flagged items is included in Appendix H of *IBM VS COBOL for OS/VS*.

CMPR2 and FLAGMIG compiler options

Important

The CMPR2/NOCMPR2 compiler option is no longer supported in Enterprise COBOL. Programs compiled with Enterprise COBOL behave as if NOCMPR2 is always in effect.

To identify OS/VS COBOL and VS COBOL II Release 2 statements that are either not supported or changed in Enterprise COBOL, use the FLAGMIG compiler option together with the CMPR2 compiler option and a previous release of a COBOL compiler, such as IBM COBOL. By compiling existing application programs with CMPR2 and FLAGMIG using an older compiler, you can identify some of the source language that needs modification in order to compile with Enterprise COBOL.

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

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.

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. For details on running existing OS/VS COBOL programs using the Language Environment runtime library, see Chapter 6, “Moving from the OS/VS COBOL run time,” on page 61. To compile OS/VS COBOL applications with Report Writer statements, you must continue to use the OS/VS COBOL compiler.

OS/VS COBOL Report Writer programs will not run above the 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 263

Language elements that are no longer 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. Convert any ISAM files to virtual storage access method/keyed sequential data set (VSAM/KSDS) files.

ISAM file handling language items affected

The following ISAM language items are no longer 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

Automated conversion options: Two conversion tools can help you convert ISAM files to VSAM/KSDS files. You can use either IDCAMS REPRO or CCCA, depending on the design of your application. The IDCAMS REPRO facility will perform the conversion unless the file has a hardware dependency.

The COBOL conversion tool (CCCA) can automatically convert the file definition and I/O statements from your ISAM COBOL language to VSAM/KSDS COBOL language. The CCCA conversion tool is described in Appendix C, "Conversion tools for source programs," on page 263.

Manual conversion actions: If the design of your application makes it impossible to convert to VSAM, you can restructure the application to separate the ISAM

statements into an I/O program that can be compiled by the OS/VS COBOL compiler. You can then separate the rest of the application logic into programs that can be upgraded to Enterprise COBOL.

You can then run your application consisting of both OS/VS COBOL programs and Enterprise COBOL programs under Language Environment. For details on running existing programs under Language Environment, see:
Chapter 6, "Moving from the OS/VS COBOL run time," on page 61
Chapter 18, "Adding Enterprise COBOL programs to existing COBOL applications," on page 219

Rewrite!: This method is presented only as a short-term migration path. Because OS/VS COBOL is no longer in service, you should rewrite your programs to eliminate the dependency on OS/VS COBOL and ISAM. (If you do not mind using an unsupported compiler, this method can still work for you.)

~ **z/OS 1.7:** ISAM support is not available with z/OS 1.7. The manual conversion
~ discussed in this section will not work if you are running on z/OS 1.7 or later.

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 no longer 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 263.

Non-IBM tools are also available to convert a BDAM file to a VSAM/RRDS file. For details, see "Vendor products" on page 270.

Manual conversion actions: If the design of your application makes it impossible to convert to VSAM, you can restructure the application to separate the BDAM statements into an I/O program that can be compiled by the OS/VS COBOL compiler. You can then separate the rest of the application logic into programs that can be upgraded to Enterprise COBOL.

After this separation is complete, your application consisting of both OS/VS COBOL programs and Enterprise COBOL programs will run under Language Environment. For details on running existing programs under Language Environment, see:

- Chapter 6, "Moving from the OS/VS COBOL run time," on page 61
- Chapter 18, "Adding Enterprise COBOL programs to existing COBOL applications," on page 219

Note: This method is presented only as a short-term migration path. Because OS/VS COBOL is no longer in service, you should rewrite your programs to eliminate the dependency on OS/VS COBOL and BDAM. (If you do not mind using an unsupported compiler, this method can still work for you.)

Communication feature

The Communication feature is not supported by 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 *IBM VS COBOL for OS/VS*, and *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:

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

An example of one way to change it, keeping the two-digit year format, is as follows:

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

01  ACCEPT-DATE.
    02 YEAR              PIC X(02).
    02 MONTH-OF-YEAR     PIC X(02).
    02 DAY-OF-MONTH      PIC X(02).
  ⋮
    ACCEPT ACCEPT-DATE FROM DATE.
    MOVE  CORRESPONDING ACCEPT-DATE TO DATE-IN-PROGRAM.
```

An example of how to change it and specify a four-digit year is as follows:

```
01  DATE-IN-PROGRAM.
    02 MONTH-OF-YEAR     PIC X(02).
    02 FILLER            PIC X(01)  VALUE "/".
    02 DAY-OF-MONTH      PIC X(02).
    02 FILLER            PIC X(01)  VALUE "/".
    02 YEAR              PIC X(04).

01  CURRENT-DATE.
    02 YEAR              PIC X(04).
    02 MONTH-OF-YEAR     PIC X(02).
    02 DAY-OF-MONTH      PIC X(02).
  ⋮
    MOVE FUNCTION CURRENT-DATE(1:8) TO CURRENT-DATE.
    MOVE CORRESPONDING CURRENT-DATE TO DATE-IN-PROGRAM.
```

EXAMINE statement

OS/VS COBOL accepted the EXAMINE statement; Enterprise COBOL does not.

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

```
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/VS 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/VS 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/VS 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).
                                  77  DAT1-CMP PIC X(8).
                                  77  DAT2-CMP PIC X(8).
        .                                 .
        .                                 .
        .                                 .
 EXHIBIT CHANGED DAT-1 DAT-2      IF DAT-1 NOT EQUAL TO DAT1-CMP
                                    DISPLAY DAT-1
                                  END-IF
                                  IF DAT-2 NOT EQUAL TO DAT2-CMP
                                    DISPLAY DAT-2
                                  END-IF
                                  MOVE DAT-1 TO DAT1-CMP
                                  MOVE DAT-2 TO DAT2-CMP
```

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

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

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

FILE-LIMIT clause of the FILE-CONTROL paragraph

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

To receive similar functions with Enterprise COBOL, you must:

- Remove references to TOTALING or TOTALED from the LABEL RECORDS clause of the FD. Define equivalent fields in WORKING-STORAGE.

  In the equivalent fields, save any key or record data (for TOTALED AREA) or update and save any record count (for TOTALING AREA) before each WRITE to the file (this reflects the last record actually written to particular volume). Do *not* save this counter in the record area of the FD.
- Include “RESERVE 1 AREA” in the SELECT clause.
- Allow for only one record per block in either the FD or the overriding JCL.
- Include:

  ```
  DECLARATIVES.
          USE AFTER STANDARD ENDING FILE (or REEL)
           LABEL PROCEDURE ON filename.
  ```

  Then, either format labels for OUTPUT files or save any data from INPUT file labels (AFTER STANDARD BEGINNING).
- Specify the NOAWO compiler option.

- Do not specify the DD DCB option OPTCD=T. It is not supported and results are unpredictable.

NOTE statement

OS/VS COBOL accepted the NOTE statement. 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. In Enterprise COBOL, if the following do not match, an OPEN statement in your program might not execute successfully:

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

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

To prevent common file status 39 problems, see Appendix H, "Preventing file status 39 for QSAM files," on page 305.

OPEN statement failing for VSAM files (file status 39)

In OS/VS COBOL, the RECORDSIZE defined in your VSAM files associated with IDCAMS was not required to match your COBOL program. In Enterprise COBOL they must match. The following rules apply to VSAM ESDS, KSDS, RRDS, and VRRDS file definitions:

Table 28. Rules for VSAM file definitions

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

OPEN statement with the LEAVE, REREAD, and DISP phrases

OS/VS COBOL allowed the OPEN statement with the LEAVE, REREAD and DISP phrases. 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.

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(ALL,SYM) 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.

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:

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

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

```
MOVE FUNCTION CURRENT-DATE (9:6) TO TIME-IN-PROGRAM
```

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

TRANSFORM statement

OS/VS COBOL supported the TRANSFORM statement. 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.

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  AND ( < C 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 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.

Dynamic CALL statements when running on CICS

Although dynamic CALL statements from OS/VS COBOL programs were never supported when running under CICS, OS/VS COBOL did not diagnose programs that issued them.

If your OS/VS COBOL programs issue dynamic CALL statements when running on CICS under Language Environment, the programs will abend with abend code U3504.

Enterprise COBOL programs can use dynamic CALL statements to call other Enterprise COBOL programs or even PL/I and C/C++ programs under CICS.

Flow of control, no terminating statement

In OS/VS COBOL, it would be possible to link-edit an assembler program to the end of an OS/VS COBOL program and have the flow of control go from the end of the COBOL program to the assembler program.

In 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/VS 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/VS 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.

MOVE statement—warning message for numeric truncation

OS/VS COBOL issued a warning message for a MOVE statement with a numeric receiver that would result in a loss of digits. For example:

```
77 A PIC 999.
77 B PIC 99.
⋮
   MOVE A TO B.
```

VS COBOL II, COBOL for MVS & VM, and COBOL for OS/390 & VM, Version 2 Release 1 do not issue a warning message for this case.

COBOL for OS/390 & VM, Version 2 Release 2 and Enterprise COBOL issue a warning message if the new compiler option DIAGTRUNC is in effect.

OCCURS clause

OS/VS 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/VS COBOL:

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

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

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

OPEN REVERSED statement

OS/VS COBOL accepted the REVERSED phrase for multireel files; 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 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.

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/VS 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/VS 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/VS COBOL compiler accepted the UNTIL and TIMES phrases in the SORT verb, for example:

```
SORT FILE-1
 ON ASCENDING KEY AKEY-1
  INPUT PROCEDURE IPROC-1
   OUTPUT PROCEDURE OPROC-1
    UNTIL AKEY-1 = 99.
```

```
SORT FILE-2
 ON ASCENDING KEY AKEY-2
  INPUT PROCEDURE IPROC-2
   OUTPUT PROCEDURE OPROC-2
    10 TIMES.
```

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/VS 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/VS COBOL, a MOVE to the SD buffer before the first RETURN in a SORT or MERGE output PROCEDURE did not overlay the data of the first record.

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/VS 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/VS 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

The following OS/VS COBOL language elements are changed in Enterprise COBOL in order to conform to the COBOL 85 Standard. For some elements, the syntax of the language is different. For others, the language syntax is unchanged, but the execution results can be different (semantics have changed).

For each element listed, there is a brief description pointing out the differences in results and what actions to take. 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 clause changes—ALPHABET keyword

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

In Enterprise COBOL, the ALPHABET keyword 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.

Note: 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 LANGLVL(1) allowed COPY statements to be preceded by an 01-level indicator, which would result in the 01-level name replacing the 01-level name in the COPY member. For example, with the following contents of COPY member MBR-A:

```
01 RECORD-A.
   05 FIELD-A...
   05 FIELD-B...
```

and a COPY statement like this:

```
01  RECORD1 COPY MBR-A.
```

the resultant source would look like this:

```
01  RECORD1.
    05 FIELD-A...
    05 FIELD-B...
```

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 LANGLVL(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. In addition, Enterprise COBOL does not accept the character G for programs that use DBCS data items.

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

Dynamic 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 29.
- For VSAM files, see Table 30 on page 141.

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

Table 29. Status key values—QSAM files

OS/VS	Enterprise COBOL	Meaning
(undefined)	04	Wrong length record; successful completion

Table 29. Status key values—QSAM files (continued)

OS/VS	Enterprise COBOL	Meaning
(undefined)	05	Optional file not present; successful completion
(undefined)	07	NO REWIND/REEL/UNIT/FOR REMOVAL specified for OPEN or CLOSE, but file not on a reel/unit medium; successful completion
00	00	Successful completion
10	10	At END (no next logical record); successful completion
30	30	Permanent error
34	34	Permanent error file boundary violation
90	90	Other errors with no further information
90	35	Nonoptional file not present
90	37	Device type conflict
90	39	Conflict of fixed file attributes; OPEN fails
90	96	No file identification (no DD statement for the file)
92	38	OPEN attempted for file closed WITH LOCK
92	41	OPEN attempted for a file in OPEN mode
92	42	CLOSE attempted for a file not in OPEN mode
92	43	REWRITE attempted when last I/O statement was not READ
92	44	Attempt to rewrite a sequential file record with a record of a different size
92	46	Sequential READ attempted with no valid next record
92	47	READ attempted when file not in OPEN INPUT or I-O mode
92	48	WRITE attempted when file not in OPEN OUTPUT, I-O, or EXTEND mode
00	48	WRITE attempted when file in OPEN I-O mode
92	49	DELETE or REWRITE attempted when file not in OPEN I-O mode
92	92	Logic error

Table 30. 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
00	00	Successful completion
00	04	Wrong length record; successful completion
00	05	Optional file not present; successful completion

Table 30. Status key values—VSAM files (continued)

OS/VS	Enterprise COBOL	Meaning
00	35	Nonoptional file not present. Can occur when the file is empty.
02	02	Duplicate key, and DUPLICATES specified; successful completion
10	10	At END (no next logical record); successful completion
21	21	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 present
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 LANGLVL(1), if a JUSTIFIED clause is specified together with a VALUE clause for a data description entry, the initial data is right-justified. For example:

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

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

```
bbbbFIRST
```

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

For example, after the following MOVE statement is executed:

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

RECEIVE-FIELD contains the value 123*bbb* (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(MIG) will cause processing very similar to OS/VS COBOL, 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.

For example, Enterprise COBOL will not generate a negative zero result, while OS/VS COBOL could. In addition, Enterprise COBOL will not repair invalid signs on input with NUMPROC(MIG), while OS/VS COBOL programs did inconsistent sign repair on input.

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

Optional word IS

For OS/VS COBOL programs, no MIGR message would be issued if the optional word IS immediately preceded objects within an abbreviated combined relation condition. For example:

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

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

Note: 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 takes place when an inner condition tests as TRUE:

1. The identifier/index associated with the outer condition is augmented by its current BY value.
2. The identifier/index associated with the inner condition is set to its current FROM value.

The following example illustrates the differences in execution:

```
PERFORM ABC VARYING X FROM 1 BY 1 UNTIL X > 3
            AFTER Y FROM X BY 1 UNTIL Y > 3
```

In OS/VS COBOL, ABC is executed 8 times with the following values:

```
X:  1  1  1  2  2  2  3  3
Y:  1  2  3  1  2  3  2  3
```

In 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 AREASA` in Enterprise COBOL.

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

With LANGLVL(1), using the RESERVE integer AREA or RESERVE integer AREAS format, you must specify the RESERVE clause with *integer + 1* areas to get equivalent processing under 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 245 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; 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:

- For z/OS: *Language Environment Installation and Customization*.
- For OS/390: *Language Environment for z/OS 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 31 on page 149. 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 31. 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.
ALL PROCEDURES	ALL PROCEDURES	Before execution of every nondebugging procedure in the outermost program After execution of every ALTER statement in the outermost program (except ALTER statements in declarative procedures).
file-name-n	(none)	See *IBM VS COBOL for OS/VS* for a description.
ALL REFERENCES OF identifier-n	(none)	See *IBM VS COBOL for OS/VS* for a description.
cd-name-1	(none)	See *IBM VS COBOL for OS/VS* for a description.

Subscripts out of range flagged at compile time

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 LANGLVL(1), when the DELIMITED BY ALL phrase of UNSTRING is specified, two or more contiguous occurrences of any delimiter are treated as if they were only one occurrence. As much of the first occurrence as will fit is moved into the current delimiter receiving field (if specified). Each additional occurrence is moved only if the complete occurrence will fit. For more information on the behavior of this phrase in OS/VS COBOL, see *IBM VS COBOL for OS/VS*.

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/VS COBOL with LANGLVL(1), will produce this result:

```
ID-R1  123       1D-D1  **        ID-C1  3
ID-R2  45678     1D-D2  **        ID-C2  5
ID-R3  90AB      1D-D3             ID-C3  4
```

Enterprise COBOL will produce this result:

```
ID-R1  123       1D-D1  *         ID-C1  3
ID-R2  45678     1D-D2  *         ID-C2  5
ID-R3  90AB      1D-D3             ID-C3  4
```

UPSI switches

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

For example, if a condition-name is defined in the SPECIAL-NAMES paragraph, the following are equivalent:

```
OS/VS COBOL                     Enterprise COBOL

SPECIAL-NAMES.                SPECIAL-NAMES.
    UPSI-0 IS MNUPO               UPSI-0 IS MNUPO
                                   ON STATUS IS UPSI-0-ON
                                   OFF STATUS IS UPSI-0-OFF
        .                              .
        .                              .
        .                              .
PROCEDURE DIVISION            PROCEDURE DIVISION
        .                              .
        .                              .
        .                              .
    IF UPSI-0 = 1 ...              IF UPSI-0-ON ...
    IF MNUPO = 0 ...               IF UPSI-0-OFF ...
```

VALUE clause condition names

For VALUE clause condition names, releases prior to Release 2.4 of OS/VS COBOL allowed the initialization of an alphanumeric field with a numeric value. For example:

```
01 FIELD-A.
    02   LAST-YEAR  PIC XX VALUE 87.
    02   THIS-YEAR  PIC XX VALUE 88.
    02   NEXT-YEAR  PIC XX VALUE 89.
```

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/VS COBOL support the use of the WHEN-COMPILED special register. The rules for use of the special register are the same for both compilers. However, the format of the data differs.

In OS/VS COBOL the format is:

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

In 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 receive 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	SKIP-CC	**Enterprise COBOL**
AFTER POSITIONING SKIP-CC	1	AFTER ADVANCING PAGE
AFTER POSITIONING SKIP-CC	' '	AFTER ADVANCING 1 LINE
AFTER POSITIONING SKIP-CC	0	AFTER ADVANCING 2 LINES
AFTER POSITIONING SKIP-CC	-	AFTER ADVANCING 3 LINES

Note: With 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 11. Compiling converted OS/VS COBOL programs

This chapter contains information on the following topics:
- Key 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.

Key compiler options for converted programs

Table 32 lists the compiler options that have special relevance to converted programs.

Table 32. Key compiler options for converted OS/VS COBOL programs

Compiler option	Comments
BUFSIZE	In OS/VS COBOL, the BUF option value specifies the total number of bytes reserved for buffers. In 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 OS/VS COBOL programs.
DIAGTRUNC	Use DIAGTRUNC to get numeric truncation flagging for MOVE statements. This is similar to the flagging in OS/VS COBOL.
NUMPROC(MIG)	NUMPROC(MIG) processes numeric signs in a way *similar* to, but not exactly like, OS/VS COBOL.
NUMCLS(ALT)	Use NUMCLS(ALT) if you were using the USERMOD shipped with OS/VS COBOL. With the USERMOD, characters A, B, and E (as well as C, D, and F) are considered valid numeric signs in the COBOL numeric class test. (You must also compile with NUMPROC(MIG).) For other alternatives for sign representation, see the *Enterprise COBOL Programming Guide*.
OPT(STD)	Use OPT(STD) if you have nonreferenced data items as eye-catchers or time/version stamps in WORKING-STORAGE. Use OPT(FULL) only if you do not need unused data items.
OUTDD(ddname)	Use this option to override the default ddname (SYSOUT) for SYSOUT output that goes to the system logic output unit. If the ddname is the same as the Language Environment MSGFILE ddname, the output is routed to the ddname designated for MSGFILE. If the ddname is *not* the same as the Language Environment MSGFILE ddname, the output from the DISPLAY statement is directed to the OUTDD ddname destination. If the ddname is not present at first reference, dynamic allocation will take place with the default name and attributes that are specified by Language Environment.
PGMNAME(COMPAT)	Use PGMNAME(COMPAT) to ensure that program names are processed in a manner compatible with OS/VS COBOL.
RMODE(24 or AUTO)	Use RMODE(24) or RMODE(AUTO) for Enterprise COBOL programs that are compiled with NORENT and mixed with OS/VS COBOL.

© Copyright IBM Corp. 1991, 2006

Table 32. Key compiler options for converted OS/VS COBOL programs (continued)

Compiler option	Comments
TRUNC	TRUNC controls the way arithmetic fields are truncated into binary receiving fields during MOVE and arithmetic operations. Use TRUNC(STD) if your shop used TRUNC as the default with OS/VS COBOL. Use TRUNC(OPT) if your shop uses NOTRUNC as the default with OS/VS COBOL (except for select program that require guaranteed nontruncation of binary data). For programs that require nontruncation of binary data, use TRUNC(BIN) — especially if there is a possibility that data being moved into binary data items can have a value larger than that defined by the PICTURE clause for the binary data item. **High-order digits:** In Enterprise COBOL, programs compiled with TRUNC(OPT) can give different results than OS/VS COBOL programs compiled with NOTRUNC. Mainly, programs can lose nonzero high-order digits. For statements where loss of high-order digits might take place, Enterprise COBOL issues a diagnostic message indicating that you should ensure that at least one of the following conditions are met: • 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 33 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 F, "Compiler option comparison," on page 289.

Table 33. OS/VS COBOL compiler options not supported by Enterprise COBOL

OS/VS COBOL option	Enterprise COBOL equivalent
BATCH/NOBATCH	Batch environment is always available (sequence of programs). CBL statements are always processed with Enterprise COBOL. **Note:** Enterprise COBOL considerations for sequence of programs are described in the *Enterprise COBOL Programming Guide*.
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 the 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 reated 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(NONE,SYM) 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.
VBSUM/NOVBSUM	Function is available with the VBREF compiler option.

Table 33. OS/VS COBOL compiler options not supported by Enterprise COBOL (continued)

OS/VS COBOL option	Enterprise COBOL equivalent
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 or OS/390 environments. 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 in OS/VS COBOL. You will need to update existing assembler programs that scan for date and time to the new format.

You can compile your programs with the 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 12. Upgrading VS COBOL II source programs

This chapter describes the differences between the VS COBOL II language and the Enterprise COBOL language. The information in this chapter will also help you determine which VS COBOL II programs require source modifications in order to compile with Enterprise COBOL. For example, VS COBOL II programs compiled with the CMPR2 option require source modification because Enterprise COBOL no longer supports the CMPR2/NOCMPR2 compiler option.

This chapter contains information on the following items that you will need to consider when upgrading VS COBOL II source programs to Enterprise COBOL:

- Determining which programs require upgrade before compiling with Enterprise COBOL
- Upgrading VS COBOL II programs compiled with the CMPR2 compiler option
- Three minor COBOL 85 Standard interpretation changes
- Change to ACCEPT statement from VS COBOL II, Release 3.0
- New reserved words
- Undocumented VS COBOL II extensions

Determining which programs require upgrade before compiling with Enterprise COBOL

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

- Programs compiled with the CMPR2 compiler option
- Programs compiled with VS COBOL II, Release 3.x, that have one or more of three minor COBOL 85 Standard features that were subject to COBOL 85 Standard interpretation changes
- Programs that were compiled with VS COBOL II, Release 3.0, that use ACCEPT . . . FROM CONSOLE
- Programs that use words which are now reserved in Enterprise COBOL
- Programs that have undocumented VS COBOL II extensions

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 no longer supported in Enterprise COBOL. However, Enterprise COBOL programs behave as if NOCMPR2 is always in effect. For information on language differences between CMPR2 and NOCMPR2 (COBOL 85 Standard) see Chapter 16, “Upgrading programs compiled with the CMPR2 compiler option” on page 175.

For information on tools that will help with the CMPR2 to NOCMPR2 conversion, see Appendix C, “Conversion tools for source programs,” on page 263.

© Copyright IBM Corp. 1991, 2006

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 appear in text that matches pseudo-text-1 of REPLACE statements.

Blank lines, which are interspersed in the matched text, will not appear in the output of the REPLACE statement. This 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 NOCMPR2 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 NOCMPR2:

- VS COBOL II, Release 3.x uses the actual length of the group as defined by the current value of the ODO object (the actual length of the group is set to spaces using the ODO object value).
- VS COBOL II, Release 4 and 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.

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 263

Table 35 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 245.

Table 34. New reserved words, by compiler, as compared to VS COBOL II.

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	JNIENVPTR, NATIONAL, NATIONAL-EDITED, XML, END-XML, XML-EVENT, XML-CODE, XML-TEXT, XML-NTEXT, FUNCTION-POINTER, GROUP-USAGE

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.

Chapter 13. Compiling VS COBOL II programs

This chapter contains information on the following topics:
- Key compiler options for VS COBOL II programs
- Prolog format changes

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

Key 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 on converting your source programs from CMPR2 to NOCMPR2 see "Upgrading programs compiled with the CMPR2 compiler option" on page 175. For information on other compiler options, see the *Enterprise COBOL Programming Guide*.

Compiling with Enterprise COBOL

Table 35 lists the Enterprise COBOL compiler options that have special relevance to converted programs.

Table 35. 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).
RMODE	Use RMODE(AUTO) or RMODE(24) for Enterprise COBOL NORENT programs that pass data to programs running in AMODE 24.
TEST	The syntax of the TEST option is different in Enterprise COBOL than in VS COBOL II. The TEST option now has three suboptions; instead of specifying TEST, you now can specify a hook location, symbol-table location, and the destination for symbolic debug information. TEST without any suboptions gives you TEST(ALL,SYM,NOSEPARATE). For more information on the TEST option, see the *Enterprise COBOL Programming Guide.*

Compiler options not supported in Enterprise COBOL

Table 36 on page 164 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.

© Copyright IBM Corp. 1991, 2006

Table 36. 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	Enterprise COBOL does not provide the FDUMP compiler option. For existing applications, FDUMP is mapped to the Enterprise COBOL TEST(SYM) compiler option, which can provide equivalent function and more. Language Environment generates a better formatted dump than VS COBOL II, regardless of the FDUMP option. However, the use of TEST(SYM) enables Language Environment to include the symbolic dump of information about data items in the formatted dump. For information about how to obtain the Language Environment formatted dump at abnormal termination, see *Language Environment Debugging Guide and Run-Time Messages*. If NOFDUMP is encountered, Enterprise COBOL issues a warning message because NOFDUMP is not supported.
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 *Migration Guide*, 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.
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 14. Upgrading IBM COBOL source programs

This chapter describes the differences between the IBM COBOL language and the Enterprise COBOL language. The information in this chapter 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 no longer supports the CMPR2/NOCMPR2 compiler option.

This chapter contains information on 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 COBOL programs
- New reserved words in Enterprise COBOL
- Undocumented IBM COBOL extensions

For information about upgrading programs compiled with the CMPR2 compiler option, see Chapter 16, "Migrating from CMPR2 to NOCMPR2," on page 175.

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 210

Determining which programs require upgrade before you compile with Enterprise COBOL

Your IBM COBOL programs will compile without change using the Enterprise COBOL compiler unless you have one or more of the following:

- 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

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

SOM-based object-oriented COBOL applications are no longer supported with Enterprise COBOL. OO COBOL syntax has been retargeted for Java-based object-oriented programming to facilitate interoperation of COBOL and Java.

The new 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 object-oriented COBOL language elements

© Copyright IBM Corp. 1991, 2006

that are not supported" on page 166 and "SOM-based object-oriented COBOL language elements that are changed" on page 167.

SOM-based object-oriented COBOL language elements that are not supported

This section describes the SOM-based OO COBOL language elements that are no longer supported in Enterprise COBOL. The following considerations apply when you migrate applications that use SOM-based OO COBOL to the Java-based OO COBOL syntax supported in Enterprise COBOL:

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 object-oriented COBOL language elements that are changed

This section describes the SOM-based OO COBOL language elements that have been changed in Enterprise COBOL. The following considerations apply when you migrate applications that use SOM-based OO COBOL to the Java-based OO COBOL syntax supported 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 no longer 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.

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.

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 263

For a complete list of reserved words see, Appendix B, “COBOL reserved word comparison,” on page 245.

Table 37. New reserved words, by compiler, as compared to VS COBOL II.

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

Table 37. New reserved words, by compiler, as compared to VS COBOL II. (continued)

Compiler	Reserved word
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	JNIENVPTR, NATIONAL, NATIONAL-EDITED, XML, END-XML, XML-EVENT, XML-CODE, XML-TEXT, XML-NTEXT, FUNCTION-POINTER, GROUP-USAGE

Undocumented IBM COBOL extensions

The IBM COBOL 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.

Language elements changed from IBM COBOL

You can get different results for COBOL statements that use intrinsic functions MAX and MIN with alphanumeric arguments of differing lengths with numeric receiver and numeric comparands. The functions produce variable-length results. However, the IBM COBOL compilers generate code that uses the length of the longest argument (in bytes) as the length of the results. When the longest argument is not the function result, extra bytes are included in the function result. If the result is used in a numeric context, incorrect output or a data exception can occur.

The Enterprise COBOL compiler treats the function result as the variable-length item that it is and generates code that extracts the length from the variable length cell (VLC).

Language Environment runtime considerations

The Enterprise COBOL compiler now uses the Language Environment STACK storage work area for several functions, including UPPER-CASE, LOWER-CASE, and NATIONAL-OF. 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 this line under the DSA MEMORY MAP: `DSA WILL BE ALLOCATED FOR nnnnnnnn BYTES`

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

Chapter 15. Compiling IBM COBOL programs

This chapter contains information on the following topics:

- Default compiler option changes from IBM COBOL
- Key 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.

Key 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 on converting your source programs from CMPR2 to NOCMPR2, see Chapter 16, “Migrating from CMPR2 to NOCMPR2,” on page 175.
- For information on 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 38 lists the options that affect compatibility between IBM COBOL and Enterprise COBOL.

Table 38. Key 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.

© Copyright IBM Corp. 1991, 2006

Table 38. Key compiler options for IBM COBOL programs (continued)

Compiler option	Comments
INTDATE	Use INTDATE(ANSI) to get the same results as COBOL/370, Release 1 for date intrinsic functions. Use INTDATE(LILIAN) if you store integer values and will be using other languages with the same data. INTDATE(LILIAN) will cause the date intrinsic functions to use the Language Environment start date, which is the same starting date that would be used by PL/I or C programs that use Language Environment date callable services. If integer dates are used only within a single program, such as converting Gregorian to Lilian and back to Gregorian in the same program, the setting of INTDATE is immaterial. 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.
PGMNAME	Use PGMNAME(COMPAT) to ensure that program names are processed in a manner similar to COBOL/370, Release 1.
NSYMBOL	Controls the interpretation of the ″N″ symbol used on literals and PICTURE clauses, indicating whether national or DBCS processing is assumed. NSYMBOL(DBCS) provides compatibility with previous releases of IBM COBOL and VS COBOL II.
TRUNC	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). For example, in a program compiled with TRUNC(BIN), a data item declared like this `01 X pic 9(2) binary.` correctly supported binary values from 0 through only 32767 in prior releases, but with Version 2 Release 2 now supports values of 0 through 65535. This support necessarily yields different arithmetic results than were obtained with the prior releases, if these very large unsigned binary values were inadvertently used.

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:

Table 39. 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: 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 *Migration Guide*, 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.
TYPECHK	The TYPECHK option is not available. TYPECHK requires SOM-based OO COBOL, which is not available with Enterprise COBOL.
WORD(NOOO)	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-INVOKE, INHERITS, INVOKE, LOCAL-STORAGE, METACLASS, METHOD, METHOD-ID, OBJECT, OVERRIDE, RECURSIVE, REPOSITORY, RETURNING, SELF, SUPER. The NOOO suboption of the WORD option is no longer supported in Enterprise COBOL.

Part 5. Moving programs to Enterprise COBOL

© Copyright IBM Corp. 1991, 2006

Chapter 16. Migrating from CMPR2 to NOCMPR2

This chapter describes the differences between the CMPR2/NOCMPR2 compiler option. If your COBOL 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 no longer supported in Enterprise COBOL. However, Enterprise COBOL programs behave as if NOCMPR2 is always in effect.

This chapter contains information on the following topics you will need to consider when examining your COBOL source programs that were compiled with the CMPR2 option:

- Upgrading programs compiled with the CMPR2 compiler option

Upgrading programs compiled with the CMPR2 compiler option

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

Beginning with VS COBOL II, Release 3.0, you could choose COBOL 85 Standard behavior (without the Intrinsic Function module) by using NOCMPR2, or the COBOL 74 Standard behavior by using the CMPR2 compiler option. The CMPR2 option provided COBOL 74 Standard behavior as implemented by VS COBOL II, Release 2, as well as *nonstandard* Release 2 extensions now implemented in the COBOL 85 Standard. 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 no longer available. Programs must be at the COBOL 85 Standard level to compile with Enterprise COBOL.

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

© Copyright IBM Corp. 1991, 2006

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.

Note: 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 40. 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 177
ALPHABETIC class	"ALPHABETIC class" on page 177
CALL ... ON OVERFLOW	"CALL . . . ON OVERFLOW" on page 178
Comparisons between scaled integers and nonnumerics	"Comparisons between scaled integers and nonnumerics" on page 179
COPY...REPLACING statements using non-COBOL characters	"COPY ... REPLACING statements using non-COBOL characters" on page 180
COPY statement using national extension characters	"COPY statement using national extension characters" on page 182
File status codes	"File status codes" on page 183
Implicit EXIT PROGRAM	"Implicit EXIT PROGRAM" on page 184
PERFORM return mechanism	"PERFORM return mechanism" on page 186
PERFORM...VARYING...AFTER	"PERFORM ... VARYING ... AFTER" on page 188
PICTURE clause with "A"s and "B"s	"PICTURE clause with "A"s and "B"s" on page 190
PROGRAM COLLATING SEQUENCE	"PROGRAM COLLATING SEQUENCE" on page 192
READ INTO and RETURN INTO	"READ INTO and RETURN INTO" on page 193
RECORD CONTAINS n CHARACTERS	"RECORD CONTAINS n CHARACTERS" on page 194
Reserved words	"Reserved words" on page 195
SET...TO TRUE	"SET . . . TO TRUE" on page 196
SIZE ERROR on MULTIPLY and DIVIDE	"SIZE ERROR on MULTIPLY and DIVIDE" on page 198
UNSTRING operand evaluation	"UNSTRING operand evaluation" on page 199
UPSI switches	"UPSI switches" on page 205

Table 40. Language elements different between CMPR2 and NOCMPR2 (continued)

Language element	Page
Variable-length group moves	"Variable-length group moves" on page 206

ALPHABET clause of the SPECIAL-NAMES paragraph

CMPR2

The ALPHABET clause does not include the keyword ALPHABET. In fact, ALPHABET is not a reserved word.

For example:

```
SPECIAL-NAMES.
   ALPHA-NAME IS STANDARD-1.
```

NOCMPR2

The ALPHABET clause requires the use of the keyword ALPHABET. ALPHABET is now a reserved keyword.

For example:

```
SPECIAL-NAMES.
   ALPHABET ALPHA-NAME IS STANDARD-1.
```

Messages

Compiling the program with the CMPR2 and FLAGMIG compiler options 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

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

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."

Note: Only errors that occur on the actual LOAD of the called program raise the ON OVERFLOW condition. Errors occurring after the program has been loaded and has started execution do not raise the condition.

NOCMPR2

Under NOCMPR2, the ON OVERFLOW condition exists if the program specified by the CALL statement cannot be made available for execution at that time.

NOCMPR2 implements the COBOL 85 Standard rules and defines the ON OVERFLOW condition to handle any "recoverable" condition that may prevent the called program from being made available.

Note: Only errors that occur on the actual LOAD of the called program raise the ON OVERFLOW condition. Errors occurring after the program has been loaded and started execution do not raise the condition.

Messages

Compiling the program with the CMPR2 and FLAGMIG options will cause the compiler to issue messages for all CALL statements that specify the ON OVERFLOW phrase. The following message will be issued:

IGYPS2012-W

MIGR The "ON OVERFLOW" phrase of the "CALL" statement will execute under more conditions under the "NOCMPR2" compiler option.

The following program fragment illustrates one situation that will be affected by this change:

```
PERFORM UNTIL ALL-ACCOUNTS-SETTLED
   ⋮
  CALL "SUBPROGA" USING CURRENT-ACCOUNT
    ON OVERFLOW
       CANCEL "SUBPROGB"
       CALL "SUBPROGA" USING CURRENT-ACCOUNT
       END-CALL
  END-CALL
   ⋮
  CALL "SUBPROGB" USING CURRENT-ACCOUNT
```

```
    ON OVERFLOW
       CANCEL "SUBPROGA"
       CALL "SUBPROGB" USING CURRENT-ACCOUNT
       END-CALL
  END-CALL
   ⋮
END-PERFORM
```

The assumption is that for some executions of this program, SUBPROGA and SUBPROGB 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 a program that exhibits CMPR2 behavior, 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 code (provided 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

In comparisons between nonnumeric items and numeric items (only integers are allowed), the value of the numeric item used in the comparison will differ if it is scaled.

CMPR2

Under CMPR2, the numeric or algebraic value of a scaled numeric item is used in comparison operations with nonnumeric items. In determining the algebraic value, all symbols P in the PICTURE character-string are included in the total number of digits, and zeros are used in their place.

NOCMPR2

Under NOCMPR2, the actual character representation or character value of the scaled numeric item is used in comparison operations with nonnumeric items. The character value for scaled numeric items does not include any digit positions specified with the symbol P. These digit positions are ignored and not counted in the size of the operand.

For example:

```
01  NUM    PIC 99PP  VALUE 2300.
01  ALPHA1 PIC XX    VALUE "23".
01  ALPHA2 PIC XXX   VALUE "23".
01  ALPHA3 PIC XXXX  VALUE "2300".

    IF NUM EQUAL ALPHA1 DISPLAY "ALPHA1".
```

```
    IF NUM EQUAL ALPHA2 DISPLAY "ALPHA2".
    IF NUM EQUAL ALPHA3 DISPLAY "ALPHA3".

                    CMPR2              NOCMPR2

Results             ALPHA3             ALPHA1
displayed                              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

This section describes three classes of non-COBOL characters occurring in library text or COPY ... REPLACING statements that are treated differently under NOCMPR2 than under CMPR2.

Non-COBOL characters are the EBCDIC characters outside the legal set of COBOL characters, excluding nonnumeric literals. Nonnumeric literals can contain any character within the character set of the computer.

CMPR2

Under CMPR2, library text and COPY ... REPLACING statements can contain operands consisting of non-COBOL characters.

NOCMPR2

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

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 COPYing & REPLACING     After COPYing & 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.
```

Then the following are the differences between CMPR2 and Enterprise COBOL after the COPYing and REPLACING.

```
      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.

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

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.

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

CMPR2
File status codes 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 41 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 Manual*.

Table 41. QSAM and VSAM file status codes with CMPR2 and NOCMPR2

VSAM file status codes		QSAM file status codes	
CMPR2	**NOCMPR2**	**CMPR2**	**NOCMPR2**
00	00 04 05 14 24 35 39 44	00	00 04 05 07 39 44

Table 41. QSAM and VSAM file status codes with CMPR2 and NOCMPR2 (continued)

VSAM file status codes		QSAM file status codes	
CMPR2	**NOCMPR2**	**CMPR2**	**NOCMPR2**
02	02		
10	10	10	10
21	21		
22	22		
23	23		
24	24		
30	30 39	30	30 39
		34	34
90	37 90	90	35 37 90
91	91		
92	38 41 42 43 44 47 48 49 92	92	38 41 42 43 46 47 48 49 92
93	93		
94	46		
95	39 95		
96	96		
97	97		

Implicit EXIT PROGRAM

To end a program, you must use an EXIT PROGRAM, STOP RUN, or GOBACK statement. You can use an EXIT PROGRAM for a called subprogram; you can use a STOP RUN for a main program. GOBACK, an IBM extension, can be used for either type of program.

CMPR2

Under CMPR2, if a program does not contain any of the 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 appear:

IGYPS2223-W
MIGR An implicit "EXIT PROGRAM" will be executed at the end of this program under the "NOCMPR2" compiler option.

If a program does contain a STOP RUN, GOBACK, or EXIT PROGRAM statement, and the NOOPTIMIZE compiler option is in effect, then use of the FLAGMIG compiler option will result in the following message:

IGYPS2224-W
MIGR An implicit "EXIT PROGRAM" may be executed at the end of this program under the "NOCMPR2" compiler option. Recompile with the "OPTIMIZE" and "FLAGMIG" compiler options. If no "MIGR" message about an implicit "EXIT PROGRAM" is issued then no implicit "EXIT PROGRAM" will be executed.

Upon re-compilation with the OPTIMIZE compiler option, the absence of any such messages indicates that the program will not have an implicit EXIT PROGRAM generated for it, while the presence of the following message indicates otherwise:

IGYOP3210-W
MIGR An implicit "EXIT PROGRAM" will be executed at the end of this program under the "NOCMPR2" compiler option.

Corrective action 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.

PERFORM return mechanism

When a paragraph or a range of paragraphs is executed with a PERFORM statement ("out-of-line PERFORM"), a mechanism at the end of the range of paragraphs causes control to be returned to the point just after the PERFORM statement.

Consider the following example:

```
    PERFORM A
    STOP RUN.
A.  DISPLAY "Hi".
B.  DISPLAY "there".
```

After displaying the message "Hi," compiler-generated code will cause the flow of control to return to the STOP RUN statement after performing paragraph A. Without this, control would fall through into paragraph B.

This code mechanism is reset to an initial state the first time a program is called or when a program is cancelled. Under NOCMPR2, it is also reset every time a program is called. Under CMPR2, the mechanism retains its last used state when a program is called twice in succession without having been cancelled. This can be important when the program issues an EXIT PROGRAM or GOBACK statement before all of the PERFORM statements have completed their execution.

Now consider this example:

```
    IF FIRST-TIME-CALLED THEN
      PERFORM A
      MOVE ZERO TO N
    ELSE
      SUBTRACT 1 FROM N
      GO TO A.
    GOBACK.
A.  IF N > 1 THEN
      GOBACK.
B.  DISPLAY "Processing continues...".
```

The program is passed a switch, FIRST-TIME-CALLED, which tells the program whether or not the program has been called without having been cancelled. It is also passed a variable, N.

CMPR2

When the program is called for the first time, the PERFORM statement will be executed. If the "N > 1" test succeeds, the program will return to the calling program.

However, this means that the PERFORM statement has not reached normal completion because paragraph A never returned to the point from which it was performed. The compiler-generated mechanism at the end of paragraph A is still "set" to return back to the PERFORM statement.

Thus, on the second call to the program, the ELSE path will be taken, 1 will be subtracted from N, and control will be transferred by the GO TO statement to paragraph A. However, if the test "N > 1" fails, the PERFORM mechanism is still

set. So, when the end of paragraph A is reached, instead of falling through into paragraph B, control is "returned" to the MOVE statement after the PERFORM statement.

These results might not be intended. The problem might occur whenever all of the following occur:

1. The program returns to the calling program with an EXIT PROGRAM or GOBACK statement.
2. A PERFORM statement performs a paragraph or a range of paragraphs, and those paragraphs 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 NOCMPR2, when the program is called for the first time, the PERFORM statement will be executed and control will flow to paragraph A. Then, depending on the result of the test "N > 1," the program will either immediately return to the calling program, or it will return 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
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

Note: 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

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 will appear for any alphabetic items that had been defined with the symbol B.

IGYDS1105-W
: **MIGR** A "PICTURE" clause was found consisting of symbols "A" and "B". This alphabetic item will be treated as an alphanumeric-edited item under the "NOCMPR2" compiler option.

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 appear only if there were no alphabetic items to be initialized. Thus, in the following example, ALPHA is never initialized, but no message alerts you to that fact.

```
01  GROUP1.
    05 ALPHA  PIC AABAA.
    05 BETA   PIC AAA.

INITIALIZE GROUP1 REPLACING ALPHABETIC DATA BY ALL "5".
```

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:

```
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".
```

Note: 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

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 using a different record description if the file contains multiple record descriptions and not all of the record descriptions are alphanumeric.

A file will be affected only when it is fixed format, there are multiple 01 record descriptions, and at least one of the record descriptions is a numeric or numeric-edited item.

When deciding which record description to use as the sending field for 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 appear:

IGYPS2283-W

MIGR The "INTO" phrase of the "READ" or "RETURN" statement was specified for file "file-name", which contained multiple records. A different record 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:

```
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 desired record description (the "longest" one), and, as a receiving field, the item that had been specified as the INTO item.

RECORD CONTAINS n CHARACTERS

The definition of RECORD CONTAINS n CHARACTERS affects existing programs and its behavior is different under CMPR2.

Consider the following example:

```
FD FILE1
   RECORD CONTAINS 40.
 01 F1R1 PIC X(20).
 01 F1R2 PIC X(40).


FD FILE2
   RECORD CONTAINS 20 TO 40.
 01 F2R1 PIC X(20).
 01 F2R2 PIC X(40).
```

CMPR2

Under CMPR2, FILE1 and FILE2 have variable-length records.

NOCMPR2

Under NOCMPR2, FILE1 has fixed-length records and FILE2 has variable-length records.

Message

Compiling the program with the CMPR2 and FLAGMIG options will cause the compiler to issue the following message for FILE1:

IGYPS1183-W
: **MIGR** "RECORD CONTAINS" clause with one integer specified is supported differently under the "NOCMPR2" compiler option.

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.

Reserved words

New reserved words were added to VS COBOL II, Release 3, IBM COBOL, and Enterprise COBOL. For a complete list of new reserved words, see Table 42.

Note: Reserved words added in any given release are also reserved in any subsequent releases.

Table 42. New reserved words by compiler as compared to VS COBOL II, Release 2

Compiler	Reserved word
VS COBOL II, Release 3 NOCMPR2	ALPHABET, ALPHABETIC-LOWER, ALPHABETIC-UPPER, BINARY, CLASS, COMMON, CONVERTING, DAY-OF-WEEK, DBCS, END-RECEIVE, EXTERNAL, GLOBAL, ORDER, PACKED-DECIMAL, PADDING, PURGE, REPLACE, STANDARD-2
COBOL/370, V1R1	FUNCTION, PROCEDURE-POINTER

Table 42. New reserved words by compiler as compared to VS COBOL II, Release 2 (continued)

Compiler	Reserved word
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	JNIENVPTR, NATIONAL, NATIONAL-EDITED, XML, END-XML, XML-EVENT, XML-CODE, XML-TEXT, XML-NTEXT, FUNCTION-POINTER, GROUP-USAGE

Messages

Compiling the program with CMPR2 and FLAGMIG compiler options causes the compiler to issue a message for each new reserved word.

The following is an example for the new reserved word "ALPHABET":

IGYPS0057-W
MIGR "ALPHABET" is a new COBOL reserved word under the "NOCMPR2" compiler option.

Corrective action for new reserved words:

Change the reserved word to a different, nonreserved COBOL word by adding or subtracting characters, or change the word entirely.

SET . . . TO TRUE

CMPR2

The SET ... TO TRUE statement is performed according to the rules of the MOVE statement.

NOCMPR2

Under NOCMPR2, SET ... TO TRUE follows the rules of the VALUE clause. There are three instances in which this 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 desired under NOCMPR2, the data in the VALUE clause for the 88-level item must be adjusted accordingly:

```
01 N PIC 9(3) BLANK WHEN ZERO VALUE ZERO. (Result = "000")
   88 V VALUE "   ".

SET V TO TRUE                              (Result = "   ")
MOVE ZERO TO N                             (Result = "   ")
```

PICTURE string with editing symbols

When a data item contains editing symbols in its PICTURE string, the character positions represented by those symbols will contain editing characters when data is moved into the data item. In a VALUE clause, initialization is not affected by the editing symbols. This means that the data in the VALUE clause will be placed into the item as is, and editing will not take place as it does in the MOVE statement.

Here's how it works under CMPR2:

```
01 E PIC X/X VALUE SPACE. (Result = "   ")
   88 V VALUE SPACE.

SET V TO TRUE             (Result = " / ")
MOVE SPACE TO E           (Result = " / ")
```

Here's how it works under NOCMPR2:

```
01 E PIC X/X VALUE SPACE. (Result = "   ")
   88 V VALUE SPACE.
SET V TO TRUE             (Result = "   ")

MOVE SPACE TO E           (Result = " / ")
```

If the behavior exhibited under CMPR2 is desired under NOCMPR2, the data in the VALUE clause for the 88-level item must be specified in edited form:

```
01 E PIC X/X VALUE SPACE. (Result = "   ")
   88 V VALUE " / ".

SET V TO TRUE             (Result = " / ")
MOVE SPACE TO E           (Result = " / ")
```

SIZE ERROR on MULTIPLY and DIVIDE

The COBOL '74 and '85 Standards 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 appear:

IGZ0036W
Truncation of high order digit positions occurred in program "program-name" on line number "n".

Message

A program that is potentially affected by this change will receive the following message when compiled with the CMPR2 and the FLAGMIG options:

IGYPG3204-W
MIGR The "ON SIZE ERROR" phrase will not be executed for intermediate results under the "NOCMPR2" compiler option.

Corrective action 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

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.

Note: Dependencies involving identifiers id-1, id-10, and id-11 are not affected by this change.

Messages

Most of the UNSTRING statements flagged with messages 3211 through 3214 will generate identical results. Only certain dependencies between the operands in the UNSTRING statement will generate different results.

For example, a dependency can exist between two operands (op-1 and op-2) in an UNSTRING statement in the following ways:

1. op-1 is subscripted, and the subscript value is modified by op-2:
   a. The subscript identifier is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.
   b. The subscript identifier is a variably located item, and an ODO object affecting the location of this item is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.
2. op-1 is a variable-length group item, and an ODO object affecting the length of this group is modified by op-2:
   a. The ODO object is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.
3. op-1 is a variably located item, and an ODO object affecting the location of this item is modified by op-2:
   a. The ODO object is used as a receiver in an INTO, DELIMITER IN, or COUNT IN operand.

Note: Dependencies generated by overlapping operands, or by specifying the same identifier as a DELIMITED BY operand and as one of the sending, INTO, or DELIMITER IN operands are illegal under both the COBOL 74 and 85 Standards and are not addressed here. Generally, results will be unpredictable.

Compiling the program 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 NOCMPR2.

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-ODOOBJ PIC 9 VALUE IS 5.
 02 VLG-1-GR.
  03 VLG-1-ODO PIC X OCCURS 1 TO 9 TIMES
                     DEPENDING ON VLG-1-ODOOBJ.
77 S-1 PIC X(20) VALUE IS ALL "123456789".

   UNSTRING S-1
            INTO VLG-1
            END-UNSTRING
```

IGYPS2222-W

MIGR The maximum length of receiver "vlg-1" will be used under the "NOCMPR2" compiler option.

Note: 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:

```
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-ODOOBJ PIC 9 VALUE IS 5.
 02 VLEN-DEL-2.
  03 VLEN-DEL-2-ODO PIC X OCCURS 1 TO 9 TIMES
                         DEPENDING ON VLEN-DEL-2-ODOOBJ.

77 S-1 PIC X(20) VALUE IS ALL "123456789".
77 R-1 PIC X(20) VALUE IS SPACES.
77 R-2 PIC X(20) VALUE IS SPACES.
77 SUB-5 PIC 99 VALUE IS 5.

   UNSTRING S-1
            DELIMITED BY OCC-DEL-1(SUB-5) OR VLEN-DEL-2,
```

```
           INTO R-1 DELIMITER IN OCC-DEL-1(SUB-5 + 1)
                    COUNT IN VLEN-DEL-2-ODOOBJ,
                R-2,
           END-UNSTRING
```

IGYPA3211-W

MIGR In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "DELIMITED BY" operand will be done only once under the "NOCMPR2" compiler option.

No corrections are required for items flagged with message 3211.

IGYPA3212-W

This message will be issued if one of the INTO identifiers in the UNSTRING statement is subscripted, refers to a variable-length group item, or refers to a variably located item.

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-ODOOBJ PIC 9 VALUE IS 9.
 02 ODO-R-3.
  03 FILLER PIC X OCCURS 1 TO 9 TIMES
                 DEPENDING ON R-3-ODOOBJ.

77 S-3 PIC X(20) VALUE IS "12 345  6789   .....".

   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN R-2-SUB,
                 OCC-R-2(R-2-SUB) COUNT IN R-3-ODOOBJ,
                 ODO-R-3,
            END-UNSTRING
```

IGYPA3212-W

MIGR In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "INTO" operand will be done only once under the "NOCMPR2" compiler option.

This UNSTRING statement will generate different results under CMPR2 and NOCMPR2 because the subscript of the second INTO receiver is modified by the COUNT IN receiver of the first INTO phrase. In addition, the length of the third INTO receiver is modified by the COUNT IN receiver of the second INTO phrase.

Under CMPR2, the values 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:

- If the original UNSTRING statement specified a WITH POINTER phrase, that phrase must be included in all of the changed UNSTRING statements. If the original UNSTRING statement *did not* specify a WITH POINTER phrase, that phrase must be added to all the changed UNSTRING statements, and the POINTER identifier must be initialized to 1.
- If the original UNSTRING statement specified a TALLYING IN phrase, that phrase must be included in all of the changed UNSTRING statements.
- If the original UNSTRING statement specified the ON OVERFLOW or NOT ON OVERFLOW phrases, those phrases must be included only in the last of the changed UNSTRING statements.

With these changes, the previous example becomes:

```
77 PTR PIC 99.

   MOVE 1 TO PTR
   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN R-2-SUB,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO OCC-R-2(R-2-SUB) COUNT IN R-3-ODOOBJ,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-3
            DELIMITED BY ALL SPACES,
            INTO ODO-R-3,
            WITH POINTER PTR,
            END-UNSTRING
```

IGYPA3213-W

This message will be issued if one of the DELIMITER IN identifiers in the UNSTRING statement is subscripted, refers to a variable-length group item, 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.

Note: Dependencies between identifiers in the same INTO phrase will not affect the result of the UNSTRING statement. CMPR2 behavior delays the effects of these dependencies until the next INTO phrase.

For example:

```
01 DEL.
 02 D-2-SUB PIC 9 VALUE IS 9.
 02 OCC-D-2-GR.
  03 OCC-D-2 PIC X OCCURS 9 TIMES.
 02 D-3-ODOOBJ PIC 9 VALUE IS 9.
 02 ODO-D-3.
  03 FILLER PIC X OCCURS 1 TO 9 TIMES
                DEPENDING ON D-3-ODOOBJ.

77 S-4 PIC X(20) VALUE IS "12 345  6789   .....".
77 R-1 PIC X(20) VALUE IS SPACES.
77 R-2 PIC X(20) VALUE IS SPACES.
77 R-3 PIC X(20) VALUE IS SPACES.

   UNSTRING S-4
```

```
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN D-2-SUB,
                 R-2 DELIMITER IN OCC-D-2(D-2-SUB)
                     COUNT IN D-3-ODOOBJ,
                 R-3 DELIMITER IN ODO-D-3,
            END-UNSTRING
```

IGYPA3213-W

MIGR In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "DELIMITER IN" operand will be done only once under the "NOCMPR2" compiler option.

This UNSTRING statement will generate different results under CMPR2 and NOCMPR2 because the subscript of the DELIMITER IN identifier of the second INTO phrase is modified by the COUNT IN receiver of the first INTO phrase. In addition, the length of the DELIMITER IN identifier of the third INTO phrase is modified by the COUNT IN receiver of the second INTO phrase.

With CMPR2 behavior, the values 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-ODOOBJ,
            WITH POINTER PTR,
            END-UNSTRING
   UNSTRING S-4
            DELIMITED BY ALL SPACES,
            INTO R-3 DELIMITER IN ODO-D-3,
            WITH POINTER PTR,
            END-UNSTRING
```

IGYPA3214-W

This message will be issued if one of the COUNT IN identifiers in the UNSTRING statement is subscripted or refers to a variably located item.

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.

Note: Dependencies between identifiers in the same INTO phrase will not affect the result of the UNSTRING statement; CMPR2 behavior delays the effects of these dependencies to the next INTO phrase.

For example:

```
01 C-2.
 02 C-2-SUB PIC 9 VALUE IS 9.
 02 OCC-C-2-GR.
  03 OCC-C-2 PIC 9 OCCURS 9 TIMES.

77 S-5 PIC X(20) VALUE IS "12 345  6789........".
77 R-1 PIC X(20) VALUE IS SPACES.
77 R-2 PIC X(20) VALUE IS SPACES.

   UNSTRING S-5
            DELIMITED BY ALL SPACES,
            INTO R-1 COUNT IN C-2-SUB,
                 R-2 COUNT IN OCC-C-2(C-2-SUB),
            END-UNSTRING
```

IGYPA3214-W

MIGR In this "UNSTRING" statement, the subscript or "OCCURS DEPENDING ON" calculations for the "COUNT IN" operand will be done only once under the "NOCMPR2" compiler option.

This UNSTRING statement will generate different results under CMPR2 and NOCMPR2 because the subscript of the COUNT IN identifier of the second INTO phrase is modified by the COUNT IN receiver of the first INTO phrase.

With CMPR2 behavior, 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

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

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).
  ```

Chapter 17. CICS conversion considerations for COBOL source

This chapter explains the source language considerations for programs that run under CICS. It describes the actions that you need to take for applications that use either CICS source or OS/VS COBOL source and involve the following functions:

- Key compiler options for programs that run under CICS
- Migrating from the separate CICS translator to the integrated CICS translator
- Base addressability considerations for OS/VS COBOL programs

CICS runtime considerations are included in the following chapters:

- Chapter 6, "Moving from the OS/VS COBOL run time," on page 61
- Chapter 7, "Moving from the VS COBOL II run time," on page 73

The CICS environment supports the use of Enterprise COBOL and Language Environment beginning with CICS Transaction Server, Version 1 Release 3 or later. You can compile and execute CICS command-level application programs. (CICS macro-level programs will neither compile with Enterprise COBOL, nor execute with Language Environment. The CICS Application Migration Aid can help with this conversion. For details, see "CICS Application Migration Aid" on page 269.)

Future considerations

OS/VS COBOL programs will be more difficult to use with CICS in the future. Starting with CICS TS Version 2.2, you can no longer use the CICS Translator for OS/VS COBOL programs. Therefore, if a program needs to be changed, it must be translated with an older version of CICS Translator or be upgraded to Enterprise COBOL.

Starting with CICS TS Version 3, you will not be able to run OS/VS COBOL programs under CICS even in Language Environment, because OS/VS COBOL programs that run under CICS require special support from the CICS product and from Language Environment. In CICS TS Version 3, this special support will not be available.

Any OS/VS COBOL programs that run under CICS should be upgraded to Enterprise COBOL.

Key compiler options for programs that run under CICS

Table 43 on page 210 lists the key compiler options for Enterprise COBOL programs that run under CICS.

© Copyright IBM Corp. 1991, 2006

Table 43. Key compiler options for programs that run under CICS

Compiler options	Comments
CICS	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. The CICS option requires that the LIB, NODYNAM, and RENT options also are in effect. Enterprise COBOL forces on these options if NOLIB, DYNAM, or NORENT are specified at the same level as the CICS option. The CICS option requires the CICS translator COBOL3; COBOL2 is not supported. There can be some problems moving from COBOL2 to COBOL3; see the APARs PQ84313 and PQ90355 for details. The programs that are affected are programs that have a literal coded as the length of the commarea, such as `LENGTH('4096')`. If NOCICS option is specified, any CICS statements found in the source program will be discarded.
DATA	Use DATA(24) in the following two cases: • For batch programs that use the CICS shared database support (DFHDRP), any parameters passed to CBLTDLI must be below the 16-MB line. • For CICS on-line programs that access a local DL/I database using CALL CBLTDLI, the parameter list and all storage areas that are referred to in the parameter list of the DL/I call must reside below the 16-MB line. For CICS/ESA you must specify TASKDATALOC(BELOW) in the TRANSACTION definition for any transaction that involves CICS online programs that access a local DL/I database by using CALL CBLTDLI. **Note:** DATA(24) is not required when you are using DBCTL. For details, see the *CICS-IMS Database Control Guide for CICS/ESA*.
DBCS	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.
LIB	LIB is required for programs translated by the CICS translator.
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 (Link PackAarea) 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.
TRUNC	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. 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).

Migrating from the separate CICS translator to the integrated translator

Many OS/VS COBOL CICS programs must be changed in order to compile them with Enterprise COBOL. The CCCA tool can make all of these changes automatically, or you can make the changes yourself using the information in this section. For more information on CCCA, see Appendix C, “Conversion tools for source programs,” on page 263.

The following considerations apply 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(NONE,SYM), XOPTS(LINKAGE,SEQ,SP)
  ```

  For the integrated CICS translator it must be changed to this:

  ```
  CBL TEST(NONE,SYM), CICS('LINKAGE,SEQ,SP')
  ```
- Do not use SIZE(MAX) when you compile programs that contain CICS statements. Storage must be left in the user region for the translator services.
- 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.

Key compiler options for the integrated CICS translator

Table 44 lists compiler options for Enterprise COBOL programs that use the integrated CICS translator.

Table 44. Key compiler options for the integrated CICS translator

Compiler option	Comments
CICS	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. The CICS option requires that the LIB, NODYNAM, and RENT options also are in effect. Enterprise COBOL forces on these options if NOLIB, DYNAM, or NORENT are specified at the same level as the CICS option. If NOCICS option is specified, any CICS statements found in the source program will be discarded.
LIB	LIB is required for programs translated by the CICS translator.
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.

Base addressability considerations for OS/VS COBOL programs

With OS/VS COBOL, you must maintain addressability to storage areas not contained within the WORKING-STORAGE SECTION of the COBOL CICS program. To satisfy program requests, an OS/VS COBOL program must keep track of storage area addresses allocated by CICS to manipulate Base locator for LINKAGE-SECTION (BLL) cells within the application program.

In Enterprise COBOL, and the associated support within CICS, this manipulation is no longer necessary. Therefore, when you upgrade from OS/VS COBOL to Enterprise COBOL, you must convert such programs. (If the COBOL CICS program does not manipulate its BLL cells addressing, conversion is not necessary.)

Note: You can do this automatically using the CCCA, as explained in Appendix C, "Conversion tools for source programs," on page 263.

The three areas that require change for programs that manipulate BLL cells are:
- SERVICE RELOAD statements
- LENGTH OF special register
- Programs using BLL cells

SERVICE RELOAD statements

In OS/VS COBOL, for programs to be executed under CICS, the SERVICE RELOAD statement is required to ensure addressability of items defined in the LINKAGE SECTION.

In Enterprise COBOL, the SERVICE RELOAD statement is not required. If it is encountered, it is treated as a comment.

LENGTH OF special register

Through the LENGTH OF special register, you no longer need to pass explicit length arguments on many of the CICS commands where length is required. You can use the LENGTH OF special register in COBOL statements as if it were an explicitly defined numeric data item. You can obtain information about the number of characters that a data item occupies in the program.

Programs using BLL cells

The following steps summarize the conversion of an OS/VS COBOL CICS program to an Enterprise COBOL CICS program. For more information, see *CICS/ESA Application Programming Guide*.

1. Remove all SERVICE RELOAD statements. The Enterprise COBOL compiler treats these statements as comments. (Although desirable, removal is not absolutely necessary.)
2. Remove all operations dealing with addressing structures in the LINKAGE SECTION greater than 4K bytes in size. A typical statement is:

   ```
   ADD +4096 D-PTR1 GIVING D-PTR2.
   ```
3. Remove all program code that assists in addressing COMMAREAs greater than 4K in size.
4. Remove redundant assignments and labels that OS/VS COBOL uses to ensure that CICS programs are correctly optimized. (This is good programming practice, but it is not essential.)

   Redundant assignments and labels include:

   - Artificial paragraph names, ones that use BLL cells to address chained storage areas
   - Artificial assignments from the object of an OCCURS DEPENDING ON clause to itself
5. Change every SET(P) option in the CICS commands to SET(ADDRESS OF L), where “L” is the LINKAGE SECTION structure that corresponds to the “P” BLL cell.
6. Specify REDEFINES clauses in the LINKAGE SECTION, if multiple record formats are defined through the SET option.
7. Review programs that use basic mapping support (BMS) data structures in their LINKAGE SECTION (check for maps that are not defined as STORAGE=AUTO). Move any maps that are not defined as STORAGE=AUTO to the WORKING-STORAGE section and remove any associated EXEC CICS GETMAIN commands.

DL/I call interface

You can convert your programs to avoid the use of BLL cells for DL/I calls as well. To do so, review your programs and make the following changes:

1. Remove BLL cells for addressing the user interface block (UIB) and program communication blocks (PCBs).
2. In the LINKAGE SECTION, retain the DLIUIB declaration and at least one PCB declaration.
3. Change the PCB call to specify the UIB directly, as follows:

   ```
   CALL "CBLTDLI" USING PCB-CALL,
                        PSB-NAME,
                        ADDRESS OF DLIUIB
   ```

4. Obtain the address of the required PCB from the address list in the UIB.

The following example shows how to get the address of the PCBs into PCB1-ADDR and PCB2-ADDR.

```
 LINKAGE SECTION.
     COPY DLIUIB.
 01  OVERLAY-DLIUIB REDEFINES DLIUIB.
     02  PCBADDR USAGE IS POINTER.
     02  FILLER  PIC XX.

 01  PCB-ADDRESSES.
     02  PCB1-ADDR    USAGE IS POINTER.
     02  PCB2-ADDR    USAGE IS POINTER.

* VACATION PCB
 01  VAC-PCB.
 COPY RDLICVP.

* HOTEL PCB
 01 HTL-PCB.
 COPY RDLICHT.
⋮
 PROCEDURE DIVISION.
⋮
     CALL 'CBLTDLI' USING PCB
                          PSBNAME
                          ADDRESS OF DLIUIB

     IF UIBFCTR IS NOT EQUAL TO LOW-VALUES
         MOVE 'PCB CALL FAILED.' TO MSG-TEXT
         PERFORM WRITE-MESSAGE-AND-AMEND
     ELSE
         SET ADDRESS OF PCB-ADDRESSES TO PCBADDR
         SET ADDRESS OF HTL-PCB TO PCB1-ADDR
         SET ADDRESS OF VAC-PCB TO PCB2-ADDR
     END-IF
⋮
```

Note: In the code above, lines ″SET ADDRESS OF PCB-ADDRESSES TO PCBADDR″ through ″SET ADDRESS OF VAC-PCB TO PCB2-ADDR″ are the commands that obtain the addresses of the required PCB from the address list in the UIB.

Example 1: Receiving a communications area

In this example, a COBOL program defines a record area within the LINKAGE SECTION. This technique is used in a number of COBOL CICS programs that define structures outside the WORKING-STORAGE SECTION.

During conversion, do the following steps:

1. Remove the address list definition defining the BLL cells; with Enterprise COBOL, BLL cells are no longer explicitly defined in the LINKAGE SECTION.
2. In SET option of the CICS command, use the ADDRESS OF special register when referring to the storage area, instead of specifying the BLL cell name.

OS/VS COBOL

```
LINKAGE SECTION.
01  PARAMETER-LIST.
  05  PARM-FILLER      PIC S9(8) COMP.
  05  PARM-AREA1-PTR   PIC S9(8) COMP.
  05  PARM-AREA2-PTR   PIC S9(8) COMP.
01  AREA1.
   05  AREA1-DATA        PIC X(100).
01  AREA2.
   05  AREA2-DATA        PIC X(100).
         .
         .
  PROCEDURE DIVISION.
         .
         .
    EXEC CICS READ DATASET("INFILE")
       RIDFLD(INFILE-KEY)
       SET(PARM-AREA1-PTR)
       LENGTH(RECORD-LEN)
    SERVICE RELOAD PARM-AREA1-PTR.
```

Enterprise COBOL

```
 LINKAGE SECTION.


01  AREA1.
       05  AREA1-DATA    PIC X(100).
01  AREA2.
       05  AREA2-DATA    PIC X(100).
         .
         .
       PROCEDURE DIVISION.
         .
         .
       EXEC CICS READ DATASET("INFILE")
           RIDFLD(INFILE-KEY)
       SET(ADDRESS OF AREA1)
           LENGTH(RECORD-LEN).
```

Example 2: Processing storage areas that exceed 4K

In OS/VS COBOL, if a LINKAGE SECTION area was greater than 4096 bytes in length, then you would have to include statements to provide addressability to the entire storage area. For Enterprise COBOL programs, these statements are no longer necessary.

The following example shows the coding for both an OS/VS COBOL program and an Enterprise COBOL program.

During conversion, do the following steps:

1. Remove the following statements used to maintain addressability:

   ```
   ADD +4096 TO RECORD-POINTER ...
   SERVICE RELOAD ...
   ```

2. Change the SET option of the CICS command from an intermediate BLL cell to the ADDRESS OF special register for the record.

OS/VS COBOL

```
LINKAGE SECTION.
 01 PARMLIST.
    .
    .
    05  RECORD-POINTERA  PIC S9(8) COMP.
    05  RECORD-POINTERB  PIC S9(8) COMP.
    .
    .
 01 FILE-RECORD.
    05  REC-AREA1         PIC X(2500).
    05  REC-AREA2         PIC X(2500).
    .
    .
PROCEDURE DIVISION.
    .
    .
    EXEC CICS READ DATASET("INFILE")
       RIDFLD(INFILE-KEY)
       SET(RECORD-POINTERA)
       LENGTH(RECORD-LEN)
    END-EXEC
    SERVICE RELOAD RECORD-POINTERA
    ADD +4096 TO RECORD-POINTERA
        GIVING RECORD-POINTERB
    SERVICE RELOAD RECORD-POINTERB.
```

Enterprise COBOL

```
 LINKAGE SECTION.


 01 FILE-RECORD.
       05  REC-DATA1       PIC X(2500).
       05  REC-DATA2       PIC X(2500).
           .
           .
 PROCEDURE DIVISION.
           .
           .
      EXEC CICS READ DATASET("INFILE")
           RIDFLD(INFILE-KEY)
           SET(ADDRESS OF FILE-RECORD)
           LENGTH(RECORD-LEN)
       END-EXEC
```

Example 3: Accessing chained storage areas

In an OS/VS COBOL CICS program, you would access chained storage areas by defining in the LINKAGE SECTION a storage area that contains a pointer to another storage area. You would then access the next chained area by copying the address of the next area into the associated BLL. It was also necessary to code a paragraph name following any statement that modified the contents of the BLL cell used to address the chained areas. With Enterprise COBOL, this chaining is simplified by using the SET statement to set the address of the chained area.

During conversion, do the following steps:

1. Change the code that moves the next address from within a storage area to the associated BLL cell. Perform the identical function in Enterprise COBOL by using the ADDRESS OF special register associated with the current and next storage areas.
2. If you like, remove the dummy paragraph names that follow references that change the contents of BLL cells. (This is good programming practice, but it is not essential.)

OS/VS COBOL

```
WORKING-STORAGE SECTION.
01 WSDATA-HOLD PIC X(100).
    .
    .
LINKAGE SECTION.
01 PARAMETER-LIST.
    .
    .
    05  CHAINED-POINTER PIC S9(8) COMP.
    .
    .
01 CHAINED-STORAGE.
    05  CHS-NEXT.       PIC S9(8) COMP.
    05  CHS-DATA        PIC X(100).
    .
    .
PROCEDURE DIVISION.
    .
    .
   MOVE CHS-NEXT. TO CHAINED-POINTER.

ANY-PARAGRAPH-NAME.
    MOVE CHS-DATA TO WSDATA-HOLD.
    .
    .
```

Enterprise COBOL

```
WORKING-STORAGE SECTION.
01 WSDATA-HOLD PIC X(100).
      .
      .
LINKAGE SECTION.
      .
      .
      .
      .
      .
      .
01 CHAINED-STORAGE.
    05 CHS-NEXT.    USAGE IS POINTER.
   05 CHS-DATA      PIC X(100)
      .
      .
PROCEDURE DIVISION.
      .
      .
  SET ADDRESS OF CHAINED-STORAGE TO CHS-NEXT.

 MOVE CHS-DATA TO WS-DATA-HOLD.
      .
      .
```

Example 4: Using the OCCURS DEPENDING ON clause

In OS/VS COBOL, if the LINKAGE SECTION contained the object of the OCCURS DEPENDING ON clause, when the number of occurrences of the subject of the OCCURS clause changed, you would need to reset the object of the OCCURS DEPENDING ON clause to trigger the update of the group length. With Enterprise COBOL, this resetting is unnecessary because the length of the group is calculated whenever the structure is referenced.

During conversion, in your Enterprise COBOL program, remove any code that resets the contents of the object of the OCCURS DEPENDING ON clause. (These references are no longer necessary.)

OS/VS COBOL

```
LINKAGE SECTION.
  01 PARMLIST.
     05  FILLER          PIC S9(8).
     05  RECORD-POINTER  PIC S9(8).
     .
     .
  01  VAR-RECORD.
    05  REC-OTHER-DATA   PIC X(30).
    05  REC-AMT-CNT      PIC 9(4).
    05  REC-AMT          PIC 9(5)
          OCCURS 1 TO 100 TIMES
            DEPENDING ON REC-AMT-CNT.
     .
     .
PROCEDURE DIVISION.
     .
     .
     EXEC CICS READ DATASET("INFILE")
         RIDFLD(INFILE-KEY)
         SET(RECORD-POINTER)
         LENGTH(RECORD-LEN)
     END-EXEC.
    MOVE REC-AMT-CNT TO REC-AMT-CNT.
    MOVE VAR-RECORD TO WS-RECORD-HOLD.
     .
     .
```

Enterprise COBOL

```
LINKAGE SECTION.

01 VAR-RECORD.
 05  REC-OTHER-DATA     PIC X(30).
 05  REC-AMT-CNT        PIC 9(4).
 05  REC-AMT            PIC 9(5)
       OCCURS 1 TO 100 TIMES
            DEPENDING ON REC-AMT-CNT.
 .
 .
PROCEDURE DIVISION.
 .
 .
 EXEC CICS READ DATASET("INFILE")
      RIDFLD(INFILE-KEY)
      SET(ADDRESS OF VAR-RECORD)
           LENGTH(RECORD-LEN)
 END-EXEC.
 .
MOVE VAR-RECORD TO WS-RECORD-HOLD.
 .
 .
```

Chapter 18. Adding Enterprise COBOL 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. 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

Restriction: On CICS, you cannot mix OS/VS COBOL programs and Enterprise COBOL programs in the same run unit. (EXEC CICS LINK and EXEC CICS XCTL will create a separate run unit.)

Important

After you add an Enterprise COBOL program to an existing application, that application must run under Language Environment.

This chapter includes information on the following topics:

- Applications comprised of RES programs
- Applications comprised of NORES programs
- Multiple load module considerations
- AMODE and RMODE considerations
- Runtime considerations
- Applications that contain assembler programs that call COBOL

When you begin adding Enterprise COBOL programs to your existing applications, you need to know the implications of link-editing existing applications with Language Environment. First, you must use the SCEELKED link-edit library, which impacts the remaining programs in the applications. How it affects existing applications depends on whether the application is comprised of:

- Programs compiled with RES
- Programs compiled with NORES
- Multiple load modules

Applications comprised of RES programs

When you add an Enterprise COBOL program to an application comprised of programs compiled with RES, you need to:

- Link-edit the changed module with Language Environment
- Run the application under Language Environment
- Understand the requirements for using dynamic and static CALL statements

The following sections list the requirements for using CALL statements on non-CICS and on CICS.

Adding Enterprise COBOL programs that use static CALL statements

Under both CICS and non-CICS, you can create an Enterprise COBOL program that uses static CALL statements to call a VS COBOL II program. However, when

© Copyright IBM Corp. 1991, 2006

you link-edit the load module with Language Environment, you must have the appropriate level of IGZEBST, the VS COBOL II bootstrap:

- The link-edit job must contain a REPLACE statement to replace IGZEBST for VS COBOL II programs that were compiled with the RES compiler option and link-edited with one of the following products:
  - VS COBOL II Release 4 runtime library without APAR PN74000 applied
  - Language Environment, Release 2 or Release 3 without APAR PN74011 applied

If you do not have the correct level of the IGZEBST routine, you will encounter a program check when the VS COBOL II program is called.

To have the best CALL performance, it is recommended that the link-edit job contain a REPLACE statement for the following:

- For COBOL/370 programs—the IGZCBSN bootstrap routine if the COBOL/370 programs were link-edited with Language Environment, Release 2, Release 3, or Release 4 without APAR PN74011 applied.
- For VS COBOL II programs compiled with the RES compiler option—the IGZEBST bootstrap if the VS COBOL II RES programs were link-edited with Language Environment, Release 4 without APAR PN74011 applied.

For an example of link-edit JCL or sample jobs in Language Environment library SCEESAMP, members IGZWRLKA, IGZWRLKB, and IGZWRLKC see Appendix J, “Link-edit example,” on page 311.

CALL statements on non-CICS

For CALL statements between OS/VS COBOL programs and Enterprise COBOL programs, parameters must be below the 16-MB line. The following sections explain the actions that you need to take for dynamic and static CALL statements.

Dynamic CALL statements

Parameters passed from an Enterprise COBOL program that dynamically calls an OS/VS COBOL program must be addressable by the OS/VS COBOL program. Specifying the appropriate Enterprise COBOL compiler options will ensure that the data is addressable by the OS/VS COBOL program.

For Enterprise COBOL programs compiled with RENT, specify the DATA(24) compiler option.

For Enterprise COBOL programs compiled with NORENT, specify the RMODE(24) or RMODE(AUTO) compiler option.

Static CALL statements

If you issue static CALL statements between OS/VS COBOL and Enterprise COBOL programs, thus forming a single load module, the load module must reside below the 16-MB line. The load module must be marked RMODE 24, AMODE 24.

For load modules with both Enterprise COBOL and OS/VS COBOL programs, you must override the default AMODE setting to AMODE 24 when the load module contains an Enterprise COBOL program compiled with NORENT. (For programs compiled with RENT, no action is necessary. The linkage editor automatically assigns the correct AMODE setting.) For instructions on how to override the default AMODE setting, see Appendix I, “Overriding linkage editor defaults,” on page 309.

CALL statements on CICS

With Enterprise COBOL, you can use static and dynamic CALL statements to call VS COBOL II, IBM COBOL, Enterprise COBOL, assembler, C, and PL/I programs. However, Language Environment does not support static or dynamic CALL statements between Enterprise COBOL programs and OS/VS COBOL programs (Continue to access OS/VS COBOL subroutines by EXEC CICS LINK).

General considerations

If your Enterprise COBOL program was processed by a CICS translator (either separate or integrated), then a caller of that program must pass the CICS EXEC interface block (DFHEIBLK) and the communication area (DFHCOMMAREA) as the first two parameters of the CALL statement. If your Enterprise COBOL program was not processed by a CICS translator, then you need to pass DFHEIBLK and DFHCOMMAREA only if they are explicitly coded in the called subprogram.

The CICS command translation process automatically inserts these parameters as the first two parameters on the corresponding PROCEDURE DIVISION USING statement in the subprogram.

Static CALL statements

In Enterprise COBOL, you can use the COBOL CALL statement to statically call VS COBOL II, IBM COBOL, and Enterprise COBOL, and assembler programs. For details of when static CALL statements are supported, see Table 49 on page 273. In addition, Language Environment supports ILC between PL/I and COBOL and between C and COBOL when running on CICS. For details, see the *Language Environment Writing Interlanguage Applications*.

In OS/VS COBOL, if multiple COBOL programs are separately compiled and then link-edited together, only the first program can contain CICS statements. With Enterprise COBOL, this restriction is removed, giving you greater flexibility in application program design.

Dynamic CALL statements

In Enterprise COBOL, you can use the COBOL CALL statement to dynamically call VS COBOL II, IBM COBOL, and Enterprise COBOL, and assembler programs. For details of when dynamic CALL statements are supported, see Table 49 on page 273. For example, programs that are the targets of dynamic CALL statements can contain CICS statements. In addition, Language Environment supports ILC between PL/I and COBOL and between C and COBOL when running on CICS. For details, see the *Language Environment Writing Interlanguage Applications*.

For Enterprise COBOL programs running under CICS, the ON EXCEPTION/OVERFLOW clause of the CALL statement is enabled.

Applications comprised of NORES programs

When you add an Enterprise COBOL program to a load module comprised of programs compiled with NORES, you need to:

- Link-edit the Enterprise COBOL program with Language Environment
- Link-edit all other programs in the application with Language Environment and understand how the behavior of these NORES programs changes when they are link-edited with Language Environment
- Link-edit REPLACE all the IGZ and ILBO CSECTs with the copies from Language Environment. For an example of link-edit JCL that shows how to

replace the current library routines in a load module with the Language Environment library routines, see Appendix J, "Link-edit example," on page 311.

Behavior before link-editing with Language Environment

Link-editing with Language Environment an application with all NORES programs is not required unless you add an Enterprise COBOL program.

Programs that are compiled with NORES and are not link-edited with Language Environment are not affected by many of the topics previously mentioned in Chapter 6, "Moving from the OS/VS COBOL run time," on page 61 and Chapter 7, "Moving from the VS COBOL II run time," on page 73. This is because programs compiled NORES do not access the Language Environment library, but continue to run in the environment in which they were link-edited.

Behavior after link-editing with Language Environment

After you add an Enterprise COBOL program and link-edit the remaining programs with Language Environment, NORES programs now behave like RES programs. Many of the considerations for programs that were compiled with RES now apply to these NORES programs that you have link-edited with Language Environment. For details, see "Implications of becoming RES-like" on page 104.

Link-edit override requirement

For load modules with both Enterprise COBOL and OS/VS COBOL programs, you must override the default AMODE setting to AMODE 24 when the load module contains an Enterprise COBOL program that was compiled with NORENT. For instructions on how to override the default AMODE setting, see Appendix I, "Overriding linkage editor defaults," on page 309.

Multiple load module considerations

Applications with multiple load modules might not be supported under Language Environment. To determine whether a multiple load module application is supported, you need to know:

- The main module
- The main program of the main module
- The submodules

Note: When you link-edit an OS/VS COBOL NORES program or a VS COBOL II NORES program that is part of a multiprogram load module with Language Environment, the COBOL library routines in the load module must be replaced with the Language Environment library routines. Failure to do so can cause unpredictable results. For a coding example of how to replace the library routines, see Appendix J, "Link-edit example," on page 311.

OS/VS COBOL considerations

Table 45 on page 223 lists all the possible combinations of multiple load modules for OS/VS COBOL programs. In the following table, if the load module has multiple programs, the main program is listed first.

Table 45. Support for applications with multiple load modules—OS/VS COBOL

Main module	Submodule	Support for	Link-edit entire application with LanEnv required
OS/VS COBOL NORES [1]	Enterprise COBOL only	No	n/a
	Enterprise COBOL and IBM COBOL	No	n/a
	Enterprise COBOL and IBM COBOL or VS COBOL II RES only	No	n/a
	OS/VS COBOL RES only	No	n/a
	OS/VS COBOL NORES only	Yes	No[2]
	Enterprise COBOL and OS/VS COBOL RES	No	n/a
	Enterprise COBOL and OS/VS COBOL NORES	No	n/a
	OS/VS COBOL RES and OS/VS COBOL NORES	No	n/a[4]
OS/VS COBOL RES	All combinations	Yes	Yes[3]
Enterprise COBOL	All combinations	Yes	Yes[3]
Enterprise COBOL and OS/VS COBOL RES	All combinations	Yes	Yes[3]
OS/VS COBOL RES and Enterprise COBOL	All combinations	Yes	Yes[3]
Enterprise COBOL and OS/VS COBOL NORES	All combinations	Yes	Yes[3]
OS/VS COBOL NORES and Enterprise COBOL	All combinations	Yes	Yes[3]
OS/VS COBOL RES and OS/VS COBOL NORES	All combinations	No[4]	n/a
OS/VS COBOL NORES and OS/VS COBOL RES	All combinations	No[4]	n/a
Enterprise COBOL and OS/VS COBOL RES and OS/VS COBOL NORES	All combinations	Yes	Yes[3]
OS/VS COBOL RES and OS/VS COBOL NORES and Enterprise COBOL	All combinations	Yes	Yes[3]
OS/VS COBOL NORES and OS/VS COBOL RES and Enterprise COBOL	All combinations	Yes	Yes[3]

Table 45. Support for applications with multiple load modules—OS/VS COBOL (continued)

Main module	Submodule	Support for	Link-edit entire application with LanEnv required

Note:

1. A load module that contains only OS/VS COBOL NORES programs can access submodules only if it uses an assembler program to load or link to the submodule.
2. ~ Existing OS/VS COBOL programs that are compiled with NORES and that do not use ISAM run without change and provide the same results as before. You do not need to link-edit these programs with Language Environment; however, you will not be able to get IBM service support for these NORES applications unless you link-edit them with Language Environment.
   ~ **ISAM:** Support was dropped by DFSMS with z/OS 1.7.
3. Link-editing with Language Environment is not required when the submodule contains only OS/VS COBOL RES programs.
4. Load modules that contain OS/VS COBOL programs compiled with RES and OS/VS COBOL programs compiled with NORES are not supported unless you include an Enterprise COBOL program or certain CSECTs. For details, see "Applications with COBOL programs compiled with RES and NORES" on page 62.

All combinations is synonymous with the different combinations of programs that the submodule can consist of (as listed next to OS/VS COBOL NORES at the beginning of the table).

VS COBOL II considerations

Applications with multiple load modules including VS COBOL II programs are supported without link-editing with Language Environment except when there is a combination of VS COBOL II NORES programs and a RES program (VS COBOL II, IBM COBOL, or Enterprise COBOL). If a combination of a VS COBOL II NORES program and an Enterprise COBOL or IBM COBOL RES program exists, then the application must be link-edited with Language Environment.

If the main module and submodule are VS COBOL II programs compiled with NORES, they will run without change and provide the same results as before. You do not need to link-edit these programs with Language Environment, however, you will not be able to get IBM service support for these NORES application unless you link-edit them with Language Environment. Once the modules are link-edited with Language Environment, they will be supported by IBM service.

In general, any multiple load module application that contains a VS COBOL II NORES program must be link-edited with Language Environment. If both modules contain only programs that are VS COBOL II RES, IBM COBOL, or Enterprise COBOL then the application is supported and does not need to be link-edited with Language Environment.

For additional link edit requirements, see "Adding Enterprise COBOL programs that use static CALL statements" on page 219.

AMODE and RMODE considerations

All OS/VS COBOL programs are AMODE 24 and RMODE 24. Enterprise COBOL programs are always AMODE ANY and can be either RMODE 24 or RMODE ANY. The WORKING-STORAGE data items can be either above or below the 16-MB line, based on the DATA, RENT and RMODE compiler options.

OS/VS COBOL programs can use CALL statements to call Enterprise COBOL programs without AMODE problems, because both programs can access below the 16-MB line data. When an Enterprise COBOL program uses a CALL statement to call an OS/VS COBOL program, you can get addressing exceptions if your data or parameter list is above the 16-MB line.

To avoid addressing exceptions, ensure that all of your data and parameter lists are below the 16-MB line by compiling with DATA(24) for RENT programs, or RMODE(24) or RMODE(AUTO) for NORENT programs.

Note: For Language Environment releases prior to Version 2 Release 9, a request for DATA(31) might have resulted in Language Environment acquiring the storage below the 16-MB line, so that incorrectly compiled programs could pass parameters to OS/VS COBOL programs without addressing exceptions. After moving to Language Environment ,Version 2 Release 9 or later, this incorrect use of DATA(31) would result in the expected addressing exception.

Figure 7 on page 226 shows the results of different combinations of compiler options and compilers. All CALL statements are dynamic and represented by arrows. The solid lines represent valid CALL statements; the dotted line represents a call that is not valid.

P1 and P5 are OS/VS COBOL programs, so the WORKING-STORAGE data items are included in the object module, and must be below the 16-MB line. For Enterprise COBOL programs that were compiled with the RENT option, WORKING-STORAGE data items are separate from the object module, and their location is controlled by the DATA compiler option (as with programs P2 and P3). For programs compiled with NORENT, the WORKING-STORAGE data items are included in the object module, so their location depends on the RMODE option (as with program P4).

All of these calls will work successfully, except for P3 calling P5. Because P3 was compiled with RENT and DATA(31), its WORKING-STORAGE and any parameter lists are located above the 16-MB line. This means that even if P3 is passing parameters that it received from P2, the parameter list cannot be addressed by P5, so the CALL will fail. The CALL will also fail if the parameters themselves are above the 16-MB line, such as data items in the WORKING-STORAGE SECTION of P1 and P3.

Note: Static CALL statements from AMODE 31 programs to OS/VS COBOL programs will always fail.

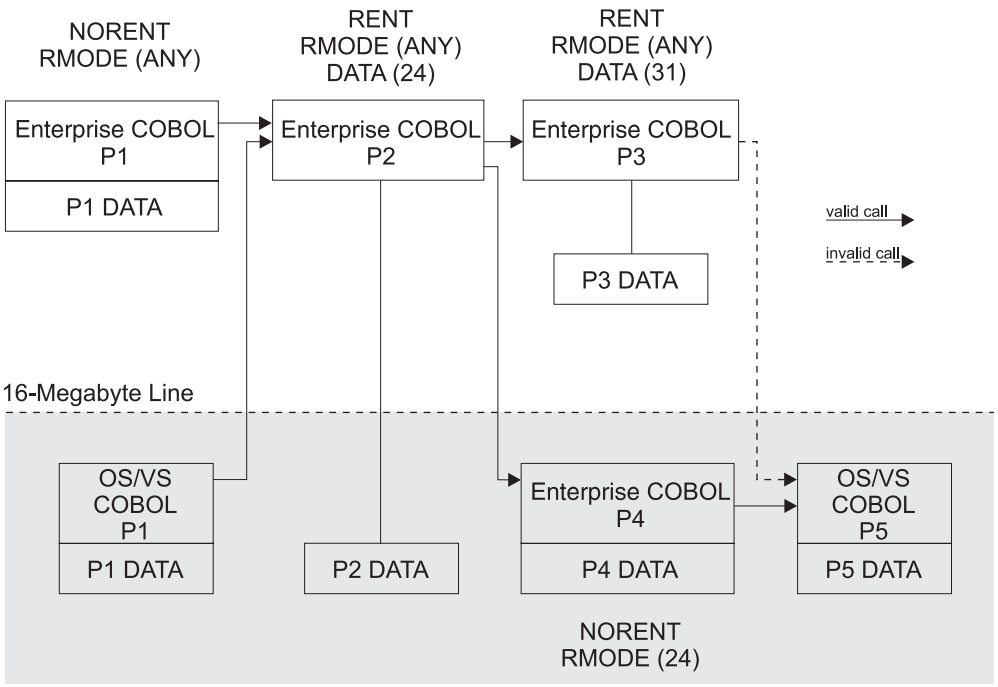


Figure 7. Valid and not valid calls between OS/VS COBOL and Enterprise COBOL programs

Runtime considerations

When adding Enterprise COBOL programs to existing applications, you have additional considerations.

ILBOSRV

You must include the Language Environment, Release 5 (or later) copy of ILBOSRV in the load module, in either of the following cases:

- If the existing load module contains and uses ILBOSTP0
- If the existing load module contains OS/VS COBOL NORES programs

TGT (Task Global Table) and RSA (Register Save Area) conventions

The TGT and RSA conventions in Enterprise COBOL, COBOL for OS/390 & VM, and COBOL for MVS & VM are different than in VS COBOL II and COBOL/370. If you have COBOL applications that are coded to assume Register 9 or Register 13 values, you must change those programs so they do not have built-in assumptions about register values.

The following will allow you to find the TGT of an Enterprise COBOL, COBOL for OS/390 & VM, or COBOL for MVS & VM program:

- When the Enterprise COBOL, COBOL for OS/390 & VM, or COBOL for MVS & VM program is running, R13 has the address of a DSA (dynamic save area) for the COBOL program. The address of the COBOL program's TGT is located at x"5C" in the DSA.
- If R13 has the address of a save area associated with a called assembler program or COBOL runtime routine, use the save area back chain to find the DSA for the COBOL program. Then add x"5C" to find the address of the TGT.

You can use the R15 slot in the previous save area to determine the entry point and signature information of the routine that owns the DSA.

Applications that contain assembler programs that call COBOL

The address of the called program must be in Register 15 when you call Enterprise COBOL programs from assembler programs.

If you have assembler programs that call COBOL programs by using BALR or BASSM and the target program address is not in Register 15 (such as `BASSM R14,R10`), then the programs will ABEND when you move from OS/VS COBOL to Enterprise COBOL.

For more information see Appendix D, "Applications with COBOL and assembler," on page 271.

Chapter 19. Upgrading programs for V3R4 SEARCH ALL

This chapter describes the actions that you need to take for certain programs that have SEARCH ALL statements and that were compiled with one of the following compilers:

- Enterprise COBOL release V3R4 prior to the installation of the PTF for APAR PK16765
- Enterprise COBOL releases V3R1 through V3R3
- COBOL for OS/390 & VM
- COBOL for MVS & VM
- COBOL/370
- VS COBOL II
- OS/VS COBOL

These actions are required due to changes in the behavior of the SEARCH ALL statement between Enterprise COBOL Version 3 Release 4 and earlier releases of COBOL. The SEARCH ALL statement is the COBOL mechanism for binary search of a table.

Enterprise COBOL Version 3 Release 4 corrects a number of errors in the implementation of the SEARCH ALL statement. SEARCH ALL statements in earlier releases of COBOL contained errors in the key comparison logic. This caused different results than might have been intended. In particular, the comparison did not give the same result as an IF statement or a sequential SEARCH statement.

First 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)` with a content of "ABCDEF" would incorrectly match a table or array key of `05 MY-KEY PIC X(4)` with a value of "ABCD". A search argument containing "ABCD␣␣" (where ␣ is a blank) would match, as expected.

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

Second 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:

© Copyright IBM Corp. 1991, 2006

Enterprise COBOL V3R4 corrects these problems with the comparison of search arguments and keys. However, some applications compiled with earlier releases might be coded in a way that depends on the incorrect behavior. You must identify and modify these applications before you move them to the Enterprise COBOL V3R4 compiler.

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. (See "Diagnostic messages text" on page 231 for the text of the messages.)

- Compiler messages - the Enterprise COBOL V3R4 compiler generates the following compiler diagnostics. Whether there is an actual impact depends on the contents of the argument at runtime.
  - 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.

Handling your programs:

To move an application to Enterprise COBOL V3R4, do one of the following sets of steps:

- Act on the compiler messages.
  1. Compile your programs with Enterprise COBOL V3R4.
  2. Review any SEARCH ALL statements that are flagged with compiler messages IGYPG3188-W or IGYPG3189-W; they are potentially impacted.
- 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 where 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.
  - 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 compare with reference-modification: For example:

    ```
    WHEN MY-KEY (MY-INDEX) = ARG(1:4)
    ```

    in the alphanumeric case, or

    ```
    WHEN MY-KEY (MY-INDEX) = ARG(3:4)
    ```

in the numeric case shown above.

- For SEARCH ALL statements where 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.

  This will avoid the warning message in the future.

Diagnostic messages text

IGYPG3188-W The search argument *argument-name* in the ″WHEN″ phrase of a ″SEARCH ALL″ statement was signed, and the corresponding key *key-name* was unsigned. The argument can match a table key only if the argument value at execution time is positive.

IGYPG3189-W The search argument *argument-name* in the ″WHEN″ phrase of a ″SEARCH ALL″ statement was longer than the corresponding key *key-name*. The argument can match a table key only if the excess digit or character positions of the argument are zeros or spaces respectively, at execution time.

IGZ0193W Search argument *argument-number* in the WHEN phrase of the SEARCH ALL statement in program *program-name* on line number *line-number* was a signed item with a negative value. The corresponding table key was an unsigned item, and so the argument could never match the key in any of the table entries.

IGZ0194W Search argument *argument-number* in the WHEN phrase of the SEARCH ALL statement in program *program-name* on line number *line-number* was longer than the corresponding key. The excess digit or character positions of the argument were not zeros or spaces respectively, and so the argument could never match the key in any of the table entries.

Part 6. Appendixes

© Copyright IBM Corp. 1991, 2006

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:

- Prerequisites
- Compatibility
- Link-editing with Language Environment
- Compiling with Enterprise COBOL
- Language Environment services
- Language Environment runtime options
- Interlanguage communication
- Subsystems
- OS/390
- z/OS
- Performance
- Service

Prerequisites

Do you have to convert all macro-level COBOL programs to CICS, Version 4 or later before they can be run with Language Environment?

Yes. You can run Language Environment only in CICS, Version 4, so any programs that you want to run under Language Environment must be upgraded to CICS, Version 4 or later.

Is Language Environment required to run a program compiled with Enterprise COBOL?

Yes, Language Environment contains the library routines required to run an Enterprise COBOL compiled program. Enterprise COBOL is a compiler, and does not contain runtime library routines.

Compatibility

I have a mix of COBOL and assembler programs. Do I need to change my assembler programs to be Language Environment enabled?

No, you do not need to change your assembler programs to be Language Environment enabled. However, your assembler programs must follow S/390 save area conventions, namely:

- The first halfword of the assembler save area must be hex zero.
- The back chain address must be set to the caller's save area.
- The back chain address must be a valid 31-bit address.

Can you run DOS/VS COBOL programs under Language Environment?

Yes. You can run programs compiled with DOS/VS COBOL under IBM Language Environment for VSE/ESA. For migration information, see the *IBM COBOL for VSE/ESA Migration Guide*, GC26-8070.

© Copyright IBM Corp. 1991, 2006

Programs compiled with DOS/VS COBOL cannot run under Program Number 5688-198 (Language Environment).

Does Language Environment support both OS/VS COBOL LANGLVL(1) and LANGLVL(2) compiled programs?

Yes.

When one runs OS/VS COBOL in compatibility mode with Language Environment, are the runtime control blocks accessed the same way?

Yes. The pointers in the TGT and other control blocks can still be used to get to the control blocks for OS/VS COBOL.

However, with Enterprise COBOL, assembler programs cannot get the address of the TGT from R13.

With VS COBOL II, we have had errors where an output DD was misspelled and a temporary file was created. This causes problems when it occurs with a large file for a one-time program run. Is this still a concern with 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.

Can you place Language Environment in LNKLST/LPALST?

Yes but it should be the only runtime library for COBOL that is in LNKLST/LPALST. You can place Language Environment in LNKLST or LPALST ahead of older COBOL runtime libraries, but the older ones would be unreachable.

Before placing Language Environment in LNKLST/LPALST, test all applications that might access the Language Environment library routines from the LNKLST/LPALST. The move to Language Environment should be staged in a controlled manner.

For more details, see “Deciding how to phase Language Environment into production mode” on page 30.

Can Language Environment coexist on a system with OS/VS COBOL and VS COBOL II?

Yes, but be aware of which library is needed and used for different applications. For example, if VS COBOL II is ahead of Language Environment in the concatenation and you recompile one program from a VS COBOL II application with Enterprise COBOL, that application will no longer run until Language Environment is ahead of the VS COBOL II runtime library in the concatenation. There are duplicate names between the different product libraries, so it is important to ensure that the correct library is being accessed.

For details, see "Invoking existing applications" on page 56.

Is it easier to convert from OS/VS COBOL to Enterprise COBOL or from OS/VS COBOL to VS COBOL II?

The source conversion effort is almost the same. Moving to the Language Environment run time is slightly more difficult if you have assembler programs that use SVC LINK or condition handling with OS/VS COBOL. Because most of time is spent testing, the two different conversion paths are approximately the same.

For a summary of runtime considerations, see:
- Table 10 on page 28 for programs running under the OS/VS COBOL run time
- Table 11 on page 29 for programs running under the VS COBOL II run time

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.

Link-editing with Language Environment

When is it necessary to link-edit applications with Language Environment to run under Language Environment?

For exact link-edit requirements, see:
- "Determining which programs require link-editing" on page 62 for programs running under the OS/VS COBOL run time
- "Determining which programs require link-editing" on page 74 for programs running under the VS COBOL II run time

Under OS/VS COBOL, aren't some library routines always invoked dynamically, even if the OS/VS COBOL program is compiled with NORES? Do I need to link-edit with Language Environment in order for these library routines to be supported when running under Language Environment?

Yes, under OS/VS COBOL ILBOD01, ILBODBE, ILBOPRM, ILBOSND, ILBOSTN, and ILBOTC2 are always invoked dynamically (unless explicitly INCLUDED by link-edit). Language Environment provides support for these library routines, regardless of whether or not the program is link-edited with Language Environment.

Can OS/VS COBOL and VS COBOL II programs call Enterprise COBOL programs?

On non-CICS, any calls between OS/VS COBOL, VS COBOL II, and Enterprise COBOL are supported.

On CICS, Enterprise COBOL programs cannot call or be called by OS/VS COBOL programs. EXEC CICS LINK/XCTL must be used instead. Calls to and from VS COBOL II programs and Enterprise COBOL programs are allowed. 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 on CICS" on page 274.

Can OS/VS COBOL NORES load modules call and be called by Enterprise COBOL programs?

Enterprise COBOL load modules can call OS/VS COBOL NORES load modules, if the NORES load module has been link-edited with Language Environment. The OS/VS COBOL NORES load module must return control to the Enterprise COBOL load module.

It is possible to have a OS/VS COBOL NORES load module with "dynamic" calls (that is, using an assembler program that loads and branches) to an Enterprise COBOL program, which can then do a COBOL dynamic call to subsequent programs.

Can you convert programs selectively to Enterprise COBOL?

For non-CICS applications, yes, as long as you follow the rules for link-editing (documented throughout this book).

For CICS applications, you cannot mix OS/VS COBOL programs and Enterprise COBOL programs in the same run unit. When you convert applications containing OS/VS COBOL programs that use CALL statements and run under CICS, you must convert all of the OS/VS COBOL programs in the run unit to Enterprise COBOL, or use EXEC CICS LINK while doing selective conversion.

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.

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 14.

Do you have to compile OS/VS COBOL and VS COBOL II programs with Enterprise COBOL to run them under Language Environment?

No. Most OS/VS COBOL programs and VS COBOL II programs (and mixes of the two) will run under Language Environment without the need to upgrade to Enterprise COBOL.

For exact details on which programs must be upgraded to Enterprise COBOL, see:

- "Determining which programs require upgrading" on page 63 for programs running under the OS/VS COBOL run time
- "Determining which programs require upgrading" on page 75 for programs running under the VS COBOL II run time

What utilities or tools can assist in converting OS/VS COBOL or VS COBOL II source to 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) 5648-B05, 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.
3. The CICS application migration aid 5695-061 assists in converting CICS macro level code in OS/VS COBOL and VS COBOL II to CICS command level code.
4. The Debug Tool Load Module Analyzer can determine the language translator for each object in your load modules.
5. The Edge Portfolio Analyzer assists in taking an inventory of your existing OS/VS COBOL and VS COBOL II load module libraries.

   Note: The Edge Portfolio Analyzer is no longer sold by IBM, but you can still purchase the product from Edge directly. For more information, you can visit their Web site at: www.edge-information.com
6. 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 the COBOL 85 Standard?

Yes, Enterprise COBOL supports all required modules of the COBOL 85 Standard at the highest level defined by the Standard.

Language Environment services

Can VS COBOL II programs call Language Environment callable services?

You can use dynamic calls from VS COBOL II programs to Language Environment date and time callable services only. You cannot use dynamic calls from VS COBOL II programs to other Language Environment callable services, nor can you use static calls to any Language Environment callable services from VS COBOL II programs.

You can dynamically call the following Language Environment services from VS COBOL II programs (any release):

CEECBLDY	CEEGMT	CEESCEN
CEEDATE	CEEGMTO	CEESECI
CEEDATM	CEEISEC	CEESECS
CEEDAYS	CEELOCT	CEE3CTY
CEEDYWK	CEEQCEN	

Can users call Language Environment callable services from BAL (assembler) programs if the programs are Language Environment-conforming assembler programs?

Yes. Any Language Environment-conforming BAL routines can use Language Environment callable services. Non-Language Environment-conforming BAL

routines *cannot* use Language Environment callable services. The *Language Environment Programming Guide* describes how to make your existing BAL programs into Language Environment-conforming BAL programs, using macros supplied by IBM with the Language Environment product.

Can OS/VS COBOL programs use Language Environment callable services?

No. OS/VS COBOL programs cannot use the Language Environment callable services directly, but OS/VS COBOL programs can call an Enterprise COBOL program to place calls to the services.

Can condition handling be added to a pure OS/VS COBOL application by converting the main routine to Enterprise COBOL?

Yes, with restrictions on recovery. For details, see "Converting programs that use ESTAE/ESPIE for condition handling" on page 275.

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 cannot be initiated by COBOL, but it can be initiated by 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.

Language Environment runtime options

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.

Does OS/VS COBOL running with Language Environment use HEAP storage?

No, OS/VS COBOL WORKING-STORAGE does not use HEAP storage.

What do HEAP(KEEP) or LIBSTACK(KEEP) do? Does the KEEP suboption keep all of the HEAP or LIBSTACK storage or just the increments of extra storage that were obtained?

The KEEP suboption causes Language Environment to keep **all** of the storage obtained, including the initial and incremental amounts.

How does ERRCOUNT relate to abends? Does ERRCOUNT only count HANDLED conditions?

ERRCOUNT is a count of errors, conditions, abends, 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.

Interlanguage communication

Why aren't assembler language programs discussed under interlanguage communication in Language Environment manuals or in IBM presentations?

Assembler language is not considered a separate language by Language Environment. There is no runtime library associated with assembler programs, so there are no runtime conflicts with assemblers and other High-Level Languages (HLLs). Assembler programs can call and be called by any of the HLLs supported by Language Environment.

Can OS/VS COBOL or VS COBOL II programs call Language Environment-conforming assembler programs?

The following calls are supported:

- On non-CICS
  - Dynamic calls from OS/VS COBOL RES programs or VS COBOL II RES programs to Language Environment-conforming assembler.
  - Static calls from OS/VS COBOL RES programs or VS COBOL II RES programs to Language Environment-conforming assembler. You must specify MAIN=NO and NAB=NO on the CEEENTRY macro.
- On CICS
  - Dynamic calls from VS COBOL II to Language Environment-conforming assembler.

For a complete list of calls between COBOL and assembler (including whether they are supported or not when running under Language Environment), see "Runtime support for assembler COBOL calls on non-CICS" on page 273 and "Runtime support for assembler COBOL calls on CICS" on page 274.

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, and this compiler option is forced 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.

Do I need to specify the TRAP(OFF) runtime option when using the CBLTDLI interface under IMS?

No, TRAP(OFF) is never recommended for COBOL programs. There are some instances when you cannot use Language Environment condition handling when using CBLTDLI under IMS. However, if you specify ABTERMENC(ABEND), database rollback will be performed automatically for severe-error conditions. For details, see the *Language Environment Programming Guide*.

I am running both OS/VS COBOL and VS COBOL II programs under CICS. All of the VS COBOL II programs are AMODE 31. Do I have to run with the Language Environment runtime option ALL31(OFF) because I have OS/VS COBOL programs (which are AMODE 24)?

You can run with ALL31(ON) if all of your VS COBOL II programs are AMODE 31. Under CICS, OS/VS COBOL programs run in their own special compatibility environment and they are not affected by the Language Environment runtime options.

Note: OS/VS COBOL programs that run under CICS require special support from the CICS product as well as support from Language Environment. With CICS Transaction Server Version 3, this special support is no longer available. OS/VS COBOL programs will not run with CICS Transaction Server Version 3 even with

Language Environment as their COBOL runtime library. You must upgrade any OS/VS COBOL programs to Enterprise COBOL as soon as possible.

Is IGZEDT4 provided in Language Environment?

Yes.

z/OS

Do I have to recompile or relink my COBOL applications to move to z/OS from OS/390?

No. Moving from OS/390 to z/OS is very much like moving from one release of OS/390 to another. Your COBOL applications will run unchanged on z/OS just like they would run on OS/390, Release 10.

Does COBOL run in 64-bit z/OS?

Yes. Though COBOL does not support 64-bit addresses 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?

Enterprise COBOL performance compared to OS/VS COBOL or VS COBOL II varies, depending on the characteristics of the applications. Information on Enterprise COBOL performance is available on the Web, in the Library Section, at:

`www.ibm.com/software/awdtools/cobol/`

IBM COBOL and Enterprise COBOL performance are approximately the same. You might see improved performance in COBOL for MVS & VM and COBOL for OS/390 & VM and later for static and dynamic calls, and in COBOL for OS/390 and later for programs compiled with TRUNC(BIN).

Service

Do I need to recompile all of my programs to get IBM service support for my applications?

As long as your programs are running with a supported run time, you do not need to recompile your programs to continue to have IBM service support. For additional details, see "Service support for OS/VS COBOL and VS COBOL II programs" on page 18.

Appendix B. COBOL reserved word comparison

This appendix contains a table showing differences between OS/VS COBOL, VS COBOL II, IBM COBOL, and Enterprise COBOL reserved words. Information on source language comparison can be found in:

- Chapter 10, "Upgrading OS/VS COBOL source programs," on page 113
- Chapter 12, "Upgrading VS COBOL II source programs," on page 157
- Chapter 14, "Upgrading IBM COBOL source programs," on page 165

This list identifies the reserved words in Enterprise COBOL, IBM COBOL, VS COBOL II, and OS/VS COBOL.

Note: Reserved words for Enterprise COBOL are included in the column marked "Enterprise COBOL." New reserved words (excluding new words reserved for future development) that have been added since IBM COBOL are highlighted in **boldface** type.

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 only. It is not reserved in Version 2 Release 1 or earlier versions.
-	The word is *not* reserved in the product. (This includes obsolete reserved words that are no longer flagged.)
CDW	The word is an 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.
SYS	The word is a word with specific meaning to the operating system. It can be used only in specific contexts within the program.
UNS	The word is a COBOL 1985 Standard reserved word for a feature not supported by this compiler. 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 46. 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
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

© Copyright IBM Corp. 1991, 2006

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
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
ARITHMETIC	-	RFD	RFD	-
AS	RFD	-	-	-
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-EXOR	-	RFD	RFD	-
B-LESS	-	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	-
BINARY-CHAR	RFD	-	-	-
BINARY-DOUBLE	RFD	-	-	-
BINARY-LONG	RFD	-	-	-
BINARY-SHORT	RFD	-	-	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
BIT	RFD	RFD	RFD	-
BITS	-	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
COMMIT	-	RFD	RFD	-
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
COMP-4	X	X	X	X
COMP-5	X	X*	RFD	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
COMP-6	-	RFD	RFD	-
COMP-7	-	RFD	RFD	-
COMP-8	-	RFD	RFD	-
COMP-9	-	RFD	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	
COMPUTATIONAL-6	-	RFD	RFD	-
COMPUTATIONAL-7	-	RFD	RFD	-
COMPUTATIONAL-8	-	RFD	RFD	-
COMPUTATIONAL-9	-	RFD	RFD	-
COMPUTE	X	X	X	X
CONDITION	RFD	-	-	-
CONFIGURATION	X	X	X	X
CONNECT	-	RFD	RFD	-
CONSOLE	SYS	SYS	SYS	X
CONSTANT	RFD	-	-	-
CONTAINED	-	RFD	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
CORR-INDEX	-	-	-	X
CORRESPONDING	X	X	X	X
COUNT	X	X	X	X
CRT	RFD	-	-	-
CSP	SYS	SYS	SYS	X
CURRENCY	X	X	X	X
CURRENT	-	RFD	RFD	-
CURRENT-DATE	-	-	-	X
CURSOR	RFD	-	-	-
C01	SYS	SYS	SYS	X

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
C02	SYS	SYS	SYS	X
C03	SYS	SYS	SYS	X
C04	SYS	SYS	SYS	X
C05	SYS	SYS	SYS	X
C06	SYS	SYS	SYS	X
C07	SYS	SYS	SYS	X
C08	SYS	SYS	SYS	X
C09	SYS	SYS	SYS	X
C10	SYS	SYS	SYS	X
C11	SYS	SYS	SYS	X
C12	SYS	SYS	SYS	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	-
DB	-	RFD	RFD	-
DB-ACCESS-CONTROL-KEY	-	RFD	RFD	-
DB-DATA-NAME	-	RFD	RFD	-
DB-EXCEPTION	-	RFD	RFD	-
DB-RECORD-NAME	-	RFD	RFD	-
DB-SET-NAME	-	RFD	RFD	-
DB-STATUS	-	RFD	RFD	-
DBCS	X	X	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	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
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
DISCONNECT	-	RFD	RFD	-
DISP	-	-	-	X
DISPLAY	X	X	X	X
DISPLAY-ST	-	-	-	X
DISPLAY-1	X	X	X	-
DISPLAY-2	-	RFD	RFD	-
DISPLAY-3	-	RFD	RFD	-
DISPLAY-4	-	RFD	RFD	-
DISPLAY-5	-	RFD	RFD	-
DISPLAY-6	-	RFD	RFD	-
DISPLAY-7	-	RFD	RFD	-
DISPLAY-8	-	RFD	RFD	-
DISPLAY-9	-	RFD	RFD	-
DIVIDE	X	X	X	X
DIVISION	X	X	X	X
DOWN	X	X	X	X
DUPLICATE	-	RFD	RFD	-
DUPLICATES	X	X	X	X
DYNAMIC	X	X	X	X
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
EMPTY	-	RFD	RFD	-
ENABLE	UNS	UNS	UNS	X
END	X	X	X	X
END-ACCEPT	RFD	-	-	-
END-ADD	X	X	X	-
END-CALL	X	X	X	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
END-COMPUTE	X	X	X	-
END-DELETE	X	X	X	-
END-DISABLE	-	RFD	RFD	-
END-DISPLAY	RFD	-	-	-
END-DIVIDE	X	X	X	-
END-ENABLE	-	RFD	RFD	-
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-SEND	-	RFD	RFD	-
END-START	X	X	X	-
END-STRING	X	X	X	-
END-SUBTRACT	X	X	X	-
END-TRANSCEIVE	-	RFD	RFD	-
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
EQUAL	X	X	X	X
EQUALS	-	RFD	RFD	-
ERASE	-	RFD	RFD	-
ERROR	X	X	X	X
ESI	UNS	UNS	UNS	X
EVALUATE	X	X	X	-
EVERY	X	X	X	X

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
EXACT	-	RFD	RFD	-
EXAMINE	-	-	-	X
EXCEEDS	-	RFD	RFD	-
EXCEPTION	X	X	X	X
EXCEPTION-OBJECT	RFD	-	-	-
EXCLUSIVE	-	RFD	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
FETCH	-	RFD	RFD	-
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
FIND	-	RFD	RFD	-
FINISH	-	RFD	RFD	-
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

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
GLOBAL	X	X	X	-
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
INDEX-1	-	RFD	RFD	-
INDEX-2	-	RFD	RFD	-
INDEX-3	-	RFD	RFD	-
INDEX-4	-	RFD	RFD	-
INDEX-5	-	RFD	RFD	-
INDEX-6	-	RFD	RFD	-
INDEX-7	-	RFD	RFD	-
INDEX-8	-	RFD	RFD	-
INDEX-9	-	RFD	RFD	-
INDEXED	X	X	X	X
INDICATE	UNS	UNS	UNS	X
INHERITS	X	X	-	-
INITIAL	X	X	X	X
INITIALIZE	X	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

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
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	-
KEEP	-	RFD	RFD	-
KEY	X	X	X	X
LABEL	X	X	X	X
LAST	UNS	UNS	UNS	X
LD	-	RFD	RFD	-
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	X
LINE	X	X	X	X
LINE-COUNTER	UNS	UNS	UNS	X
LINES	X	X	X	X
LINKAGE	X	X	X	X
LOCALLY	-	RFD	RFD	-
LOCAL-STORAGE	X	X	-	-
LOCALE	RFD	-	-	-
LOCK	X	X	X	X
LOW-VALUE	X	X	X	X
LOW-VALUES	X	X	X	X
MEMBER	-	RFD	RFD	-
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	-	-	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
MODE	X	X	X	X
MODIFY	-	RFD	RFD	-
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
NOMINAL	-	-	-	X
NONE	-	RFD	RFD	-
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
ONLY	-	RFD	RFD	-
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

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
OTHER	X	X	X	-
OTHERWISE	-	-	-	X
OUTPUT	X	X	X	X
OVERFLOW	X	X	X	X
OVERRIDE	X	X	-	-
OWNER	-	RFD	RFD	-
PACKED-DECIMAL	X	X	X	-
PADDING	X	X	X	-
PAGE	X	X	X	X
PAGE-COUNTER	UNS	UNS	UNS	X
PARAGRAPH	-	RFD	RFD	-
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
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	-
PRIOR	-	RFD	RFD	-
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	-	-	-
PROTECTED	-	RFD	RFD	-
PROTOTYPE	RFD	-	-	-
PURGE	UNS	UNS	UNS	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
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
REALM	-	RFD	RFD	-
RECEIVE	UNS	UNS	UNS	X
RECONNECT	-	RFD	RFD	-
RECORD	X	X	X	X
RECORD-NAME	-	RFD	RFD	-
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
RELATION	-	RFD	RFD	-
RELATIVE	X	X	X	X
RELEASE	X	X	X	X
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
REPEATED	-	RFD	RFD	-
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

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
RERUN	X	X	X	X
RESERVE	X	X	X	X
RESET	X	X	X	X
RESUME	RFD	-	-	-
RETAINING	-	RFD	RFD	-
RETRIEVAL	-	RFD	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
ROLLBACK	-	RFD	RFD	-
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
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
SESSION-ID	-	RFD	RFD	-
SET	X	X	X	X

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
SHARED	-	RFD	RFD	-
SHARING	RFD	-	-	-
SHIFT-IN	X	X	X	-
SHIFT-OUT	X	X	X	-
SIGN	X	X	X	X
SIZE	X	X	X	X
SKIP-1	-	-	-	X
SKIP-2	-	-	-	X
SKIP-3	-	-	-	X
SKIP1	CDW	CDW	CDW	-
SKIP2	CDW	CDW	CDW	-
SKIP3	CDW	CDW	CDW	-
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	-
STANDARD-3	-	RFD	RFD	-
STANDARD-4	-	RFD	RFD	-
START	X	X	X	X
STATUS	X	X	X	X
STOP	X	X	X	X
STORE	-	RFD	RFD	-
STRING	X	X	X	X
SUB-QUEUE-1	UNS	UNS	UNS	X
SUB-QUEUE-2	UNS	UNS	UNS	X

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
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	SYS	SYS	SYS	X
SYSIPT	SYS	SYS	SYS	-
SYSLIST	SYS	SYS	SYS	X
SYSLST	SYS	SYS	SYS	-
SYSOUT	SYS	SYS	SYS	X
SYSPCH	SYS	SYS	SYS	-
SYSPUNCH	SYS	SYS	SYS	X
SYSTEM-DEFAULT	RFD	-	-	-
S01	SYS	SYS	SYS	X
S02	SYS	SYS	SYS	X
S03	SYS	SYS	SYS	-
S04	SYS	SYS	SYS	-
S05	SYS	SYS	SYS	-
TABLE	UNS	UNS	UNS	X
TALLY	X	X	X	X
TALLYING	X	X	X	X
TAPE	X	X	X	X
TENANT	-	RFD	RFD	-
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	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
TO	X	X	X	X
TOP	X	X	X	X
TOTALED	-	-	-	X
TOTALING	-	-	-	X
TRACE	X	X	X	X
TRACK-AREA	-	-	-	X
TRACK-LIMIT	-	-	-	X
TRACKS	-	-	-	X
TRAILING	X	X	X	X
TRANSCEIVE	-	RFD	RFD	-
TRANSFORM	-	-	-	X
TRUE	X	X	X	-
TYPE	X	X*	-	-
TYPEDEF	RFD	-	-	-
UNEQUAL	-	RFD	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	SYS	SYS	SYS	X
UPSI-1	SYS	SYS	SYS	X
UPSI-2	SYS	SYS	SYS	X
UPSI-3	SYS	SYS	SYS	X
UPSI-4	SYS	SYS	SYS	X
UPSI-5	SYS	SYS	SYS	X
UPSI-6	SYS	SYS	SYS	X
UPSI-7	SYS	SYS	SYS	X
USAGE	X	X	X	X
USAGE-MODE	-	RFD	RFD	-
USE	X	X	X	X
USER-DEFAULT	RFD	-	-	-
USING	X	X	X	X
VAL-STATUS	RFD	-	-	-
VALID	RFD	RFD	RFD	-
VALIDATE	RFD	RFD	RFD	-

Table 46. Reserved word comparison (continued)

Reserved word	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
VALIDATE-STATUS	RFD	-	-	-
VALUE	X	X	X	X
VALUES	X	X	X	X
VARYING	X	X	X	X
WAIT	-	RFD	RFD	-
WHEN	X	X	X	X
WHEN-COMPILED	X	X	X	X
WITH	X	X	X	X
WITHIN	-	RFD	RFD	-
WORDS	X	X	X	X
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-NTEXT	X	-	-	-
XML-TEXT	X	-	-	-
ZERO	X	X	X	X
ZEROES	X	X	X	X
ZEROS	X	X	X	X
<	X	X	X	X
<=	X	X	X	-
+	X	X	X	X
*	X	X	X	X
**	X	X	X	X
-	X	X	X	X
/	X	X	X	X
>	X	X	X	X
>=	X	X	X	-
=	X	X	X	X

Appendix C. Conversion tools for source programs

This appendix describes the conversion tools available for your assistance during the actual conversion activity. These tools are:
- MIGR compiler option (OS/VS COBOL)
- Other programs that aid conversion

This appendix helps you to determine which, if any, of the conversion tools to use, and understand how to use them and how to analyze the conversion tool output to assess the extent of the remaining conversion effort.

Note: All of these conversion tools assume that the program you are converting is a valid OS/VS COBOL, VS COBOL II, or IBM COBOL program; that is, it is a program that is written according to the rules given in *IBM VS COBOL for OS/VS*, *VS COBOL II Application Programming Language Reference*, and *COBOL for OS/390 & VM Language Reference*, with no undocumented extensions.

MIGR compiler option

You can use the OS/VS COBOL MIGR compiler option whenever you are planning to convert an OS/VS COBOL program to Enterprise COBOL. This option helps you understand the magnitude of the conversion effort. It can also ease any planned future conversion by helping you avoid using OS/VS COBOL source language not supported by Enterprise COBOL. By compiling your programs using MIGR, you can determine ahead of time what language elements must be converted.

There are incompatibilities in the following areas:
- New reserved words 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 no longer supported

You can set the MIGR compiler option either as an installation default at install time, or when compiling an OS/VS COBOL program. When you set MIGR on, the compiler flags most statements that are changed in or not supported by 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

© Copyright IBM Corp. 1991, 2006

- 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 . . .
- 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 OCCURS DEPENDING ON clause
- Changes in intermediate results for ON SIZE ERROR option
- 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

Following are 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 LANGLVL(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 LANGLVL(2) statements not supported

The following OS/VS COBOL statements, applicable to both the LANGLVL(1) and LANGLVL(2) compiler options, are not supported in 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 LANGLVL sensitive, but is flagged only under Communications.)

Report Writer:

- INITIATE, GENERATE, and TERMINATE verbs
- LINE-COUNTER, PAGE-COUNTER, and PRINT-SWITCH special registers
- Nonnumeric literal IS mnemonic-name in SPECIAL NAMES
- REPORT clause of FD
- REPORT SECTION header
- USE BEFORE REPORTING declarative

Note: The Report Writer Precompiler can convert these statements for you. See "COBOL Report Writer Precompiler" on page 269.

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:

- On the workstation
  - Report Writer
- On the host:
  - COBOL and CICS/VS Command Level Conversion Aid (CCCA)

- CICS application migration aid
- COBOL Report Writer Precompiler
- Vendor products

Report Writer for OS/2 and Windows

Report Writer for OS/2 and Windows, product 5801-AAR part number 4226060, delivers a general-purpose COBOL printing facility and provides the familiar host Report Writer function on the workstation.

For additional information, see "COBOL Report Writer Precompiler" on page 269.

WebSphere Studio Asset Analyzer

~ WebSphere Studio Asset Analyzer, product number 5655-M22, 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), product number 5648-B05, converts CICS and non-CICS source code into source code compilable by Enterprise COBOL. CCCA is also included in Debug Tool Utilities and Advanced Functions, Version 3, product number 5655-J18.

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 following are 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 and/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 detail, see the *COBOL and CICS/VS Command Level Conversion Aid* manual.

Note: If you need to both convert your programs and structure your code and you have COBOL/SF available, you can select an option within COBOL/SF that will activate CCCA. CCCA will convert your program, and then COBOL/SF will structure it.

When to use CCCA

If you plan to convert your applications from OS/VS COBOL or VS COBOL II to 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 47.

Statements dealing with the secondary BLLs are replaced by CONTINUE.

Table 47. COBOL statements dealing with primary BLLs

Original source	Source after conversion
MOVE BLL1 TO BLL2	SET ADDRESS OF REC2 TO ADDRESS OF REC1
MOVE ID TO BLL	MOVE ID TO LCP-WS-ADDR-COMP SET ADDRESS OF REC1 TO LCP-WS-ADDR-PNTR
MOVE BLL TO ID	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC MOVE LCP-WS-ADDR-COMP TO ID
ADD ID1, .. TO BLL	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD ID1, TO LCP-WS-ADDR-COMP SET ADDRESS OF REC TO LCP-WS-ADDR-PNTR
ADD BLL TO ID1, ID2	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD LCP-WS-ADDR-COMP TO ID1, ID2
ADD ID1, ID2 GIVING BLL	ADD ID1, ID2 GIVING LCP-WS-ADDR-COMP SET ADDRESS OF REC TO LCP-WS-ADDR-PNTR
ADD ID, BLL1 GIVING BLL2 BLL3	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD ID, LCP-WS-ADDR-COMP GIVING LCP-WS-ADDR-COMP SET ADDRESS OF REC2 TO LCP-WS-ADDR-PNTR SET ADDRESS OF REC3 TO LCP-WS-ADDR-PNTR
ADD ID1, BLL1 GIVING ID2 ID3	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD ID1, LCP-WS-ADDR-COMP GIVING ID2 ID3
SUBTRACT statements	The conversion is performed in the same way as ADD.
COMPUTE BLL = exp (BLL)	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC COMPUTE LCP-WS-ADDR-COMP = exp (LCP-WS-ADDR-COMP)
COMPUTE ID = exp (BLL)	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC COMPUTE ID = exp (LCP-WS-ADDR-COMP)
COMPUTE BLL = exp ...	COMPUTE LCP-WS-ADDR-COMP = exp ...

CICS Application Migration Aid

The CICS Application Migration Aid, Product Number 5695-061, simplifies the conversion of COBOL and assembler language application programs from macro-level application programming interface (API) to the command-level API. Simpler macros are converted automatically, providing new source code with the equivalent command-level functions. More complex macros are partially converted, and guidance information is provided to aid you in completing the conversion. The original source code is still available in all cases.

Command-level is a requirement for applications that are to run on CICS/ESA, Version 4 or later.

COBOL Report Writer Precompiler

The Report Writer Precompiler, product number 5798-DYR, has two functions. It can be used to precompile applications containing Report Writer statements so the code will be acceptable to the Enterprise COBOL compilers, or it can permanently convert Report Writer statements to valid Enterprise COBOL statements.

The Report Writer Precompiler offers the following features:

- Extended Report Writer language capabilities
- Automatic invocation of the target COBOL compiler—as though Report Writer statements in the source program are being processed by the COBOL compiler itself
- Single consolidated source listing merges information from the precompiler listing and the COBOL compiler listings
- COPY library members can contain Report Writer statements
- Supports the 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. To use the Load Module Analyzer, you must purchase and install Debug Tool Utilities and Advanced Functions Version 6.1 (5655-P15).

The Edge Portfolio Analyzer

The Edge Portfolio Analyzer helps you to take an inventory of your existing OS/VS COBOL and VS COBOL II load modules. The Edge Portfolio Analyzer can:

- Determine which version and release of the OS/VS COBOL compiler or the VS COBOL II compiler created the load module
- Determine which compiler options were specified when the load module was compiled
- Determine which load modules call for the current system date
- Determine which CSECTs need to be replaced, such as ILBOSRV.

Note: The Edge Portfolio Analyzer is no longer sold by IBM, but you can still purchase the product from Edge directly. For more information you can visit their Web site at:
www.edge-information.com

Vendor products

A number of non-IBM conversion tools are available to help you upgrade your source programs to Enterprise COBOL programs and move to Language Environment. IBM has compiled a list of vendor products enabled to work with Language Environment and Enterprise COBOL in the *Language Environment Run-Time Migration Guide*.

Appendix D. Applications with COBOL and assembler

This chapter contains information for applications that contain mixed COBOL programs and assembler programs. It includes information on:

- Determining requirements for calling and called assembler programs
- Determining which assembler/COBOL calls are supported on non-CICS
- Determining which assembler/COBOL calls are supported on CICS
- Converting programs that use ESTAE/ESPIE for condition handling
- Converting programs that change the program mask
- Upgrading applications that use an assembler driver
- Invoking a COBOL program with an MVS ATTACH
- Assembler programs that load and call COBOL programs
- Freeing storage in subpools
- Invoking programs - AMODE requirements

Some information on applications with both assembler programs and COBOL programs is included in other chapters of this book. Table 48 lists the information and page reference of where additional information on assembler programs is located.

Table 48. Additional information about assembler programs and COBOL programs

Program attribute	Reference
Assembler routine using a LINK SVC	"Determining which programs require upgrading" on page 63 or "Determining which programs require upgrading" on page 75
Assembler routine using a LINK SVC while in a reusable environment	"Using ILBOSTP0" on page 83
Assembler programs that pass procedure names	"Language elements that changed from OS/VS COBOL" on page 136
Assembler programs that do not close files under MVS	"Closing files in non-COBOL and OS/VS COBOL programs" on page 81
AMODE behavior when COBOL programs return to assembler programs	Chapter 9, "Upgrading Language Environment release levels," on page 105

Determining requirements for calling and called assembler programs

When assembler programs are either called by or call COBOL programs, they must follow the S/390 linkage convention. If not, your applications will terminate with 40XX abends.

Calling assembler programs

When the assembler program is the caller, R13 must point to the caller's register save area (18 words), and the first 2 bytes of the save area must be zero. The save area back chain must be set with a valid 31-bit address. (That is, the high-order byte must be cleared when running AMODE 24.)

The recommended way to establish a register save area address is:

© Copyright IBM Corp. 1991, 2006

```
          LA   13, SAVEAREA
          .
          .
          .
SAVEAREA  DS   18F
```

as this will ensure that the high-order byte is cleared for AMODE 24 programs.

If you must use a branch and link type of instruction to jump over the save area and set its address, the preferred method is:

```
          BAS  13,SKIP
SAVEAREA  DS   18F
SKIP      DS   0H
```

If you use a BAL instruction, you must clear the high-order byte yourself, as follows:

```
          BAL  13,SKIP
SAVEAREA  DS   18F
SKIP      LA   13,0(,13)
```

The BAL instruction puts the instruction length code, the condition code (CC), and the program mask in the high-order byte of the register. We recommend using the BAS instruction instead of the BAL instruction, since the BAS instruction places zeros in the high-order byte, thus preventing the 24-bit addressing problem.

If the program passes parameters, a parameter list must be prepared, and the address of this list loaded into R1. R1 must be set to zero if no parameter list is passed. R14 must contain the return address in the assembler program, and R15 must contain the address of the entry point of the COBOL program.

Note: If you pass a parameter list, it must be a group of one or more contiguous fullwords, each of which contain the address of a data item to be passed to the COBOL program. It is recommended that the high-order bit of the last fullword address be set to 1, to flag the end of the list.

Called assembler programs

A called assembler program must save the registers and store other information in the save area passed to it by the COBOL program. In particular, the COBOL save area must be properly back chained from the save area of an assembler program. The assembler program must also contain a return routine that:

- Loads the address of the COBOL save area back into R13
- Restores the contents of the other registers
- Optionally sets a return code in R15
- Branches to the address in R14
- 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 on non-CICS

Table 49 lists the possible combinations of calls involving COBOL programs and assembler programs and indicates whether the calls are supported or not supported when running under Language Environment under non-CICS. For the calls that are *not* supported, this table also lists the symptom (message or abend code) that is returned in most cases. In some cases, depending on the application environment, the symptom might not occur. You could receive a different failure, or the application might appear to run successfully.

Note: The term, *IBM COBOL* refers to COBOL/370, COBOL for MVS & VM and COBOL for OS/390 & VM.

Table 49. Language Environment's support for calls between COBOL programs and assembler (Asm) programs on non-CICS

Calls from			Issued to					
Program issuing	**Call type**	**Enterprise COBOL**	**IBM COBOL**	**VS COBOL II**	**OS/VS COBOL**	**LanEnv[1] Asm main**	**LanEnv[1] Asm subrtn**	**Non-LanEnv Asm**
Enterprise COBOL	Static	Yes	Yes	Yes	Yes	No[2]	Yes	Yes
IBM COBOL		Yes	Yes	Yes	Yes	No[2]	Yes	Yes
VS COBOL II		Yes	Yes	Yes	Yes	No[2]	Yes[3]	Yes
OS/VS COBOL		Yes	Yes	Yes	Yes	No[2]	Yes[3]	Yes
Enterprise COBOL	Dynamic	Yes	Yes	Yes	Yes	No[2]	Yes	Yes
IBM COBOL		Yes	Yes	Yes	Yes	No[2]	Yes	Yes
VS COBOL II		Yes	Yes	Yes	Yes	No[2]	Yes	Yes
OS/VS COBOL		Yes	Yes	Yes	Yes	No[2]	Yes	Yes
Asm (LanEnv)	VCON	Yes	Yes	Yes	Yes	No[2]	Yes	Yes
Asm (non-LanEnv)		Yes	Yes	Yes	Yes	Yes[4]	No[5]	Yes
Asm (LanEnv)	LOAD	Yes	Yes	Yes	Yes	No[2]	Yes	Yes
Asm (non-LanEnv)	BALR	Yes	Yes	Yes	Yes	Yes[4]	No[5]	Yes
Asm (LanEnv)	LINK	Yes	Yes	Yes	Yes[6]	Yes	No[5]	Yes
Asm (non-LanEnv)		Yes	Yes	Yes	Yes[6]	Yes	No[5]	Yes

Note: The failure symptoms described in these notes are as they would appear when the Language Environment TRAP(ON) and ABTERMENC(ABEND) runtime options are in effect.

1. CEEENTRY macro with MAIN=YES creates a Language Environment assembler main. If you specify MAIN=NO on the CEEENTRY macro, a Language Environment assembler subroutine is created. The default is MAIN=YES.

2. 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.
3. You must specify NAB=NO and MAIN=NO on the CEEENTRY macro. Otherwise, you will receive failure symptom 0C1, 0C4, or 0C5 abend.
4. If the non-Language Environment assembler caller is running within an established Language Environment enclave, see note 2.
5. Failure symptom of 0C1, 0C4, or 0C5 abend.
6. 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 on CICS

Table 50 lists the possible combinations of calls involving COBOL programs and assembler programs and indicates whether the calls are supported or not when running with Language Environment under CICS. For the calls that are *not* supported, this table also lists the symptom (message or abend code) that will be returned in most cases. In some cases, depending on the application environment, the symptom might not occur; you could receive a different failure, or the application might appear to run successfully.

Note: The term *IBM COBOL* refers to COBOL/370, COBOL for MVS & VM, and COBOL for OS/390 & VM.

Table 50. Language Environment's support for calls between COBOL programs and assembler (Asm) programs that run on CICS

Calls from		**Issued to**						
Call type	**Program issuing**	**Enterprise COBOL**	**IBM COBOL**	**VS COBOL II**	**OS/VS COBOL**	**LanEnv[1] Asm main**	**LanEnv[1] Asm subrtn**	**Non-LanEnv Asm**
Static	Enterprise COBOL	Yes	Yes	Yes	No[2]	No[3]	Yes	Yes
	IBM COBOL	Yes	Yes	Yes	No[2]	No[3]	No[4]	Yes
	VS COBOL II	Yes	Yes	Yes	No[2]	No[3]	No[4]	Yes
	OS/VS COBOL	No[4]	No[4]	No[4]	Yes	No[4]	No[4]	Yes
Dynamic	Enterprise COBOL	Yes	Yes	Yes	No[3]	No[3]	Yes	Yes
	IBM COBOL	Yes	Yes	Yes	No[3]	No[3]	Yes	Yes
	VS COBOL II	Yes	Yes	Yes	No[3]	No[3]	Yes	Yes
	OS/VS COBOL	No[5]	No[5]	No[5]	No[4]	No[5]	No[4]	No[5]
EXEC CICS LINK	Enterprise COBOL	Yes	Yes	Yes	Yes	No[3]	No[4]	Yes
	IBM COBOL	Yes	Yes	Yes	Yes	No[3]	No[4]	Yes
	VS COBOL II	Yes	Yes	Yes	Yes	No[3]	No[4]	Yes
	OS/VS COBOL	Yes	Yes	Yes	Yes	No[3]	No[4]	Yes
VCON	Asm (LanEnv) Asm (non-LanEnv)	Yes No[4]	Yes No[4]	No[4] No[4]	No[4] No[4]	No[3] No[3]	Yes No[4]	Yes Yes
EXEC CICS LINK	Asm (LanEnv) Asm (non-LanEnv)	Yes Yes	Yes Yes	Yes Yes	Yes Yes	No[3] No[3]	No[4] No[4]	Yes Yes

Note: The failure symptoms described in these notes are as they would appear when the Language Environment TRAP(ON) and ABTERMENC(ABEND) runtime options are in effect.

1. CEEENTRY macro with MAIN=YES creates a Language Environment assembler main. If you specify MAIN=NO on the CEEENTRY macro, a Language Environment assembler subroutine is created. The default is MAIN=YES.
2. Failure symptom of: message IGZ0079S.
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 4 program check).
5. Failure symptom of: U3504 abend.

Converting programs that use ESTAE/ESPIE for condition handling

OS/VS COBOL and VS COBOL II applications can contain user-written condition handling routines written in assembler. Normally, you would code an assembler routine (this is your ESTAE routine) to set an ESTAE and register an ESTAE exit. When an abend occurs, the ESTAE exit receives control. (Although this section references ESTAE, this information also applies to assembler routines that set ESPIEs and register ESPIE exits.)

When you run programs under Language Environment, the Language Environment condition manager receives control for errors, program interrupts, and abends. Existing user-written condition handling routines do not work under Language Environment.

Error handling routines in existing programs

For OS/VS COBOL and VS COBOL II applications, total source conversion is not necessary. You need to convert only programs that either establish or perform condition handling.

However, if the application code is maintained in OS/VS COBOL or VS COBOL II, automatic stack frame collapse will not take place. If the application resumes after the condition handler gets control, you must explicitly cancel any programs that will be reentered before they are reentered.

To convert programs that establish or perform condition handling:

1. Replace the CALL to the ESTAE routine with a CALL to CEEHDLR (a Language Environment callable service). Only Enterprise COBOL programs can use CEEHDLR.

   If you want to resume and have more control of where to resume, use the SET RESUME POINT service (CEESRP). For details, see the *Language Environment Programming Reference*.
2. Make your ESTAE exit into a separate Language Environment-conforming routine (Enterprise COBOL, Language Environment-conforming assembler, or Language Environment-conforming C).
3. Change the logic of your condition handler. You must use Language Environment services to indicate whether you want to RESUME, PERCOLATE (let another handler take control), or PROMOTE (change the condition to another condition). You can also use Language Environment services to find the name of the program that incurred the condition, or retrieve the error message associated with the condition.

Establishing user-written condition handling routines

Using Enterprise COBOL's PROCEDURE-POINTER data item (an IBM extension to the COBOL 85 Standard) in conjunction with the SET statement, you can establish your own condition handling routine using Language Environment-provided callable services. The user-written condition handling routines receive control before the Language Environment default condition handling. You can write user-written condition handling routines in either Enterprise COBOL, Language Environment-conforming C, or Language Environment-conforming assembler.

Advantages of user-written condition handling routines

Managing the point where you resume execution after handling an error is much simpler with Language Environment than with OS/VS COBOL or VS COBOL II. With OS/VS COBOL, VS COBOL II, and Language Environment, recursive calls are not allowed. For example, under OS/VS COBOL and VS COBOL II, if Program A calls Program B and an error occurs in Program B thus passing control back to Program A, Program A then cannot call Program B because this would cause a recursive error. Thus, after intercepting an error, the condition handler must resume at the next sequential instruction (NSI) following the instruction that incurred the error.

With OS/VS COBOL and VS COBOL II, to resume in a program other than the one that incurred the error, you are required to cancel the programs that are active at the time of the error and that might be reentered after the ESTAE exit received control.

When running Enterprise COBOL or Language Environment-conforming assembler programs under Language Environment, Language Environment automatically deactivates your programs when you change the resume point. Applying the above example to programs that run under Language Environment, after resuming in Program A, you *can* call Program B. Language Environment has deactivated Program B, so no recursion occurs.

For VS COBOL II programs, the move resume cursor function (Language Environment callable service CEEMRCR) will cause stack frame collapse, where programs that were active are rendered inactive, and can then be reentered (except for VS COBOL II programs compiled with NOCMPR2 that use nested programs).

For VS COBOL II programs that can be reentered, you can:

- Resume in the NSI of the failing program.
- Move the resume cursor to the instruction following the CALL statement to the failing program.
- Move the resume cursor to the NSI of the caller of the caller of the failing program.

Converting programs that change the program mask

When a VS COBOL II program calls an assembler program that changes the program mask (for example, uses an SPM instruction), the program mask is restored after the call to the assembler program.

With 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.

Calling assembler programs that expect a certain program mask

When you are running mixed COBOL and assembler programs in a application under Language Environment, called assembler programs cannot rely on a predictable program mask. You could experience ABEND 0CA (decimal overflow) in your assembler program when running such a mixed application with Language Environment. If an assembler program has certain program mask requirements, it must set the program mask to what it needs and then restore the program mask before returning.

VS COBOL II: With VS COBOL II, the program mask always had the same setting when an assembler program was called.

Language Environment sets the program mask based on the requirements of the High-Level Languages (HLLs) that are initialized in the enclave. Each HLL informs Language Environment of its program mask requirements, and Language Environment ORs all of the requirements together and sets the program mask accordingly.

Language Environment neither saves nor restores the program mask setting across calls to Language Environment services or calls within the Language Environment environment.

The program mask requirement for COBOL is to have each mask setting OFF, but the other HLLs have requirements for one or more of the program masks to be set ON. This table shows some of these requirements:

Table 51. Program mask requirement by HLL

Language	Fixed-Point Overflow mask	Decimal Overflow mask	Exponent Underflow mask	Significance mask
C/C++	OFF	ON	OFF	OFF
PL/I MVS & VM	ON	ON	ON	OFF
Enterprise PL/I	OFF	ON	ON	OFF

To support the language semantics when one or more program mask settings are ON, Language Environment uses the condition handling enablement step to determine which exceptions should be enabled (that is, treated as conditions) and which should be ignored, based on the language that is currently active on the stack.

Example scenario: When COBOL and C/C++ are both initialized, the decimal overflow mask is ON and all the other mask settings are OFF. When there is a decimal overflow in a COBOL program or a COBOL runtime routine, Language Environment condition handling gets control for the exception. During the enablement step, Language Environment determines that COBOL wants to ignore the decimal overflow and resume. Language Environment then resumes the application. If the decimal overflow occurs while an assembler program or a C routine is active, the exception is enabled; and a condition is signalled.

There are cases when both COBOL and C/C++ are initialized even though only COBOL programs are run. For example, COBOL requires the C/C++ runtime in the following cases:

- The COBOL program contains an XML PARSE or an XML GENERATE statement.

~ • The COBOL program is compiled with one of the following compiler options:
~ – DLL
~ – PGMNAME(LONGUPPER)
~ – PGMNAME(LONGMIXED)
~ • Object oriented COBOL.

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 on how to make your existing assembler programs Language Environment-conforming, see the *Language Environment Programming Guide*.

Modify the assembler driver

If you want to modify the assembler driver routine, you can replace the OS/VS COBOL ILBOSTP0 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 is not available with OS/VS COBOL.

If either IGZERRE or ILBOSTP0 is statically linked with the assembler driver, the assembler driver must be link-edited with Language Environment. If ILBOSTP0 is used, you must specify the Language Environment options ALL31(OFF) and STACK(,,BELOW).

For alternative methods to use preinitialization, see “Initializing the runtime environment” on page 94.

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. For more information, see “Recommended runtime options for non-CICS applications” on page 51.

Invoking a COBOL program with an MVS ATTACH

When you invoke a COBOL main program by using an MVS ATTACH, Language Environment processes parameter lists differently than VS COBOL II.

Under Language Environment, when a COBOL program is invoked directly by using ATTACH SVC (including the invocation of a batch program by the operating system and invocation from TSO CALL/ATTACH), the parameter list is always processed as a ″PARM=″ style.

In VS COBOL II, when a COBOL program is invoked directly by using ATTACH SVC (including the invocation of a batch program by the operating system and invocation from TSO CALL/ATTACH), the parameter list is processed as a ″PARM=″ style only when:

- Register 1 is nonzero.
- The word addressed by register 1 (the first parameter pointer word) has the end of list (EOL) bit on.
- The parameter addressed by the EOL bit is aligned on a halfword or greater boundary.

Otherwise, register 1 and the parameter list are passed without change.

There are two ways to get compatible behavior:

1. Change the main program to Language Environment-conforming assembler, and use PLIST=OS keyword in the CEEENTRY macro. Then have the assembler program call the COBOL program. The sample code below shows how to do this.

```
ASMLE3   CEEENTRY PPA=MAINPPA,AUTO=WORKSIZE,MAIN=YES,PLIST=OS
         USING WORKAREA,13
         L     15,A1C401P         Get the addr of the COBOL pgm
         BALR  14,15              Call it with parm list unchanged
*=====================================================================
*   Terminate Language Environment.
*=====================================================================
         CEETERM  RC=0
MAINPPA  CEEPPA                   Constants describing the code block
*=====================================================================
*        The Workarea and DSA
*=====================================================================
A1C401P  DC    V(A1C401P)         VCON FOR COBOL pgm
WORKAREA DSECT
         ORG    *+CEEDSASZ        Leave space for the DSA fixed part
         DS     0D
WORKSIZE EQU    *-WORKAREA
         CEEDSA                   Mapping of the Dynamic Save Area
         CEECAA                   Mapping of the Common Anchor Area
         CEEEDB                   Mapping of the Enclave Data Block
         END   ASMLE3
```

2. You can modify Language Environment to ensure that the parameter list processing for a COBOL main program invoked by using an ATTACH has the same behavior as when the COBOL program runs with the VS COBOL II run time. To modify the parameter list processing under Language Environment, run the sample customization job IGZWAPSX with a modified copy of IGZEPSX (the COBOL parameter list exit routine).

   For instructions on how to modify IGZEPSX see:
   - For OS/390 and z/OS: *Language Environment for z/OS Customization*

Assembler loading and calling COBOL programs

VS COBOL II programs use a different copy of WORKING-STORAGE for each call under the VS COBOL II run time; however, under Language Environment the same copy of WORKING-STORAGE is used for each call. These different situations occur when the following conditions are true:

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

In addition, the programs enter their initial state when run under the VS COBOL II runtime library. However, when running under Language Environment the programs are entered in their last-used state, unless there is an intervening CANCEL.

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 reLOAD 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:

- 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

Note: 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:

- 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, do one of the following:

- Have the assembler program statically call a COBOL program that performs the dynamic call and performs the CANCEL.

- Under Language Environment, Release 7 (OS/390, Release 3) or later, use the CEEFETCH and the CEERELES macros.

Note: CEEFETCH can be used to load Language Environment-conforming programs only. CEEFETCH can be used to load IBM COBOL and Enterprise COBOL programs. CEEFETCH cannot be used to load VS COBOL II or OS/VS COBOL programs.

Freeing storage in subpools (z/OS and OS/390 only)

On z/OS and OS/390, Language Environment allocates storage out of subpool 1 and subpool 2. VS COBOL II only used subpool 0. If you free storage in subpool 1 or subpool 2, you can lose data placed in those subpools by Language Environment.

Invoking programs - AMODE requirements

With VS COBOL II, assembler programs could invoke COBOL programs regardless of the AMODE specification. For example:

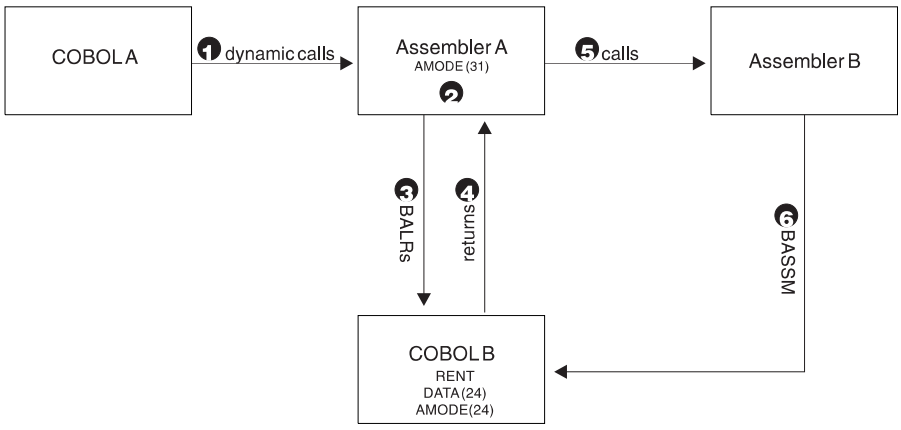


Figure 8. Effect of AMODE for invoking programs

In the figure above, the following occurs:

1. Program `COBOLA` dynamically calls `AssemblerA`. (Assembler is AMODE 31.)
2. `AssemblerA` loads `COBOLB` which is RENT, DATA(24), and AMODE(24).
3. `AssemblerA` BALRs to `COBOLB`. (Now, COBOLB is entered in AMODE 31.)
4. `COBOLB` returns to `AssemblerA`.
5. `AssemblerA` calls `AssemblerB`.
6. `AssemblerB` does a BASSM to `COBOLB`. (COBOLB is entered in AMODE 24.)
7. COBOLB abends because it expects to be entered in AMODE 31.

Under Language Environment, COBOL programs compiled with VS COBOL II, IBM COBOL, or Enterprise COBOL that are invoked by assembler programs must be entered in the same AMODE each time that they are called.

To avoid the abend in the example above:

- In Step 2, `AssemblerA` must save the AMODE.
- In Step 3, `AssemblerA` needs to BASSM to `COBOLB` instead of BALR.
- In Step 4, `AssemblerA` needs to restore the AMODE saved in Step 2.

If an assembler program that is AMODE 31 calls a COBOL program that is AMODE 24, the assembler program must also be RMODE 24 in order for COBOL to return to the assembler program. If the assembler program is AMODE ANY in this case, an abend might result upon return from the COBOL program as a result of branching to an address that is not valid. This is because R14 will contain a 31-bit address from the assembler program's save area, but COBOL will return to the assembler program in AMODE 24.

Appendix E. Debugging tool comparison

Debug Tool is a program analyzer that runs within Language Environment and supports a number of high-level languages, including Enterprise COBOL.

For Enterprise COBOL, Debug Tool is orderable as a feature of the compiler.

Debugging existing applications

Debug Tool does not support debugging of OS/VS COBOL programs.

Debug Tool provides support to VS COBOL II, Release 3.0 and higher if the VS COBOL II programs have been compiled with the TEST compiler option. Considerations for debugging VS COBOL II programs include:

COBTEST
COBTEST can still be used to debug VS COBOL II programs that use the VS COBOL II library. COBTEST is incompatible with Debug Tool, and cannot be used with Language Environment.

Test scripts
If you use COBTEST to run program tests and have a library of COBTEST test scripts, Debug Tool contains a command translator to aid in your conversion efforts.

Debugging migrated applications

When you compile your OS/VS COBOL, VS COBOL II, or IBM COBOL programs with Enterprise COBOL, you need to know which compiler options to use to enable debugging.

Applications with OS/VS COBOL programs

Debug Tool can be used to debug a run unit that contains a combination of Enterprise COBOL and OS/VS COBOL programs. If you want to convert your OS/VS COBOL programs, use the Convert and Compile option in the Program Preparation feature that is provided in Debug Tool Utilities and Advanced Functions.

If a run unit contains OS/VS COBOL programs that specify the COUNT compiler option or expect the FLOW runtime option, you cannot use Debug Tool to debug *any* programs in the run unit (including VS COBOL II, IBM COBOL, and Enterprise COBOL programs).

Applications with VS COBOL II programs

FDUMP compiler option
The FDUMP compiler option is not supported in Enterprise COBOL, but its function is. FDUMP is mapped to the TEST(SYM) compiler option.

TEST compiler option
The TEST compiler option syntax has been modified to allow you to specify whether and where compiled-in hooks should be located, and whether dictionary and CALC tables should be generated. Five hook-location

© Copyright IBM Corp. 1991, 2006

suboptions are possible: ALL, NONE, STMT, PATH, and BLOCK. Two symbol-table suboptions are possible: SYM and NOSYM.

When you compile with the TEST(...,SYM,...) option, the compiler generates additional symbolic information for the enhanced debugging capabilities that Version 3 of Debug Tool provides. This additional information might increase the size of the application load module if you also use TEST(...,SYM,NOSEPARATE). To reduce the size of a load module, specify the TEST(...,SYM,SEPARATE) suboption.

In an application that mixes VS COBOL II programs and Enterprise COBOL programs, Debug Tool can only debug the VS COBOL II programs that are compiled with TEST.

Batch debugging considerations

Debug Tool provides a function equivalent to the batch mode of COBTEST, along with some additional features.

In batch mode, Debug Tool gets its input from an input file and Debug Tool output is written to an output file similar to COBTEST batch mode. The differences between Debug Tool batch mode and COBTEST batch mode are that the COBTEST commands, RECORD, QUALIFY, RESTART, GO, and RUN, behave differently in batch mode than in interactive mode. With Debug Tool, all commands behave the same in both modes.

Some Debug Tool commands, such as fullscreen commands, are not allowed in batch mode.

Initiating Debug Tool

The way in which you initiate Debug Tool is different for Enterprise COBOL programs. When you use Debug Tool, the application program starts first and the Language Environment runtime TEST option controls the invocation of Debug Tool. (With VS COBOL II, you are required to invoke COBTEST first; it then starts the application to be debugged.)

You can also invoke 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 on the Language Environment TEST option subparameters and suboptions, see *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 *Language Environment Programming Reference*.

Command language comparison

Table 52 compares the command language of TESTCOB, COBTEST, and Debug Tool. As you can see, many of Debug Tool commands are different from the commands used in COBTEST and TESTCOB. In most cases, the function is identical.

Commands not supported by a specific debugger are indicated with a dash.

Table 52. Debug Tool command language comparison

TESTCOB command	COBTEST command	Debug Tool command	Function
AT	AT	AT	Set a breakpoint
-	AUTO[1]	MONITOR LIST	Automatically monitor variables
-	COLOR[1]	PANEL COLORS	Display panel to set color attributes
-	DOWN[2]	WINDOW DOWN	Move window down
DROP	DROP	CLEAR	Delete a defined symbol
DUMP	DUMP	CALL %DUMP	Produce a memory dump
END	QUIT	QUIT	End the debug session
EQUATE	EQUATE	SET EQUATE or CLEAR EQUATE	Define a symbol
-	FLOW	LIST LAST	Collect control flow information
-	FREQ	SET FREQUENCY	Tally the execution counts for verbs
GO	GO	GO	Start execution of the COBOL program
HELP (TSO only)	HELP[3]	HELP	Provide online help information
IF	IF	IF	Evaluate a condition
-	LEFT[2]	WINDOW LEFT	Move window left
-	LINK	LOAD or CALL[5]	Set up a LINKAGE SECTION
LIST	LIST	LIST, DESCRIBE[6]	List contents of variables
LISTBRKS	LISTBRKS	LIST AT	List breakpoints in effect
-	LISTEQ	QUERY EQUATES	List the defined symbols
LISTFILE	LIST	DESCRIBE ATTRIBUTES	List attributes of a file
-	LISTFREQ	LIST FREQUENCY	List frequency counts of verbs

Table 52. Debug Tool command language comparison (continued)

TESTCOB command	COBTEST command	Debug Tool command	Function
-	LISTINGS[1]	PANEL LISTINGS	Display panel associating program with a listing
-	MOVECURS[1]	CURSOR	Move cursor from command line to a window or vice versa
NEXT	NEXT	STEP	Set temporary breakpoint at next verb
-	NORECORD	SET LOG OFF	Turn off debug session logging
OFF	OFF	CLEAR AT	Turn off a breakpoint in effect
OFFWN	OFFWN	CLEAR AT	Turn off a conditional breakpoint
-	ONABEND	AT OCCURRENCE CEExxxx	Perform an action at occurrence of a Language Environment condition, where xxxx is the condition returned from Language Environment
-	PEEK	QUERY prefix	Show breakpoints within a line
-	POSITION[1]	SCROLL TO	Move to a certain location of a displayed object
-	PREVDISP[1]	-[6]	Show the last displayed user ISPF screen
-	PRINTDD	-[7]	Route the output to a data set
-	PROC	AT CALL	Trap calls to certain subprograms
-	PROFILE[1]	PANEL PROFILE	Display panel to set user profile attributes
QUALIFY	QUALIFY	SET QUALIFY	Change the current program qualify
-	RECORD	SET LOG ON	Turn on logging of debug session
-	RESTART	RESTART[8]	Reinitialize program without exiting debugger
-	RESTORE[1]	SET or QUALIFY RESET	Return to current point of execution
-	RIGHT[2]	WINDOW RIGHT	Move window right
RUN	RUN	CLEAR AT then GO	Remove breakpoints and run to completion
-	SEARCH[1]	FIND	Search for a string in displayed object
-	SELECT	SHOW	Display a specific frequency count
SET	SET	MOVE or SET or COMPUTE	Alter contents of a variable
SOURCE	SOURCE[9]	WINDOW OPEN	Display source program statements
-	STEP	STEP	Execute specified number of verbs
-	SUFFIX[1]	SET SUFFIX	Turn on visual display of frequency
TRACE	TRACE	AT GLOBAL STATEMENT or AT GLOBAL PATH	Trace flow of execution
-	UP[2]	WINDOW UP	Move window up
-	VTRACE[1]	STEP	Dynamic visual trace of program
WHEN	WHEN	AT * IF...	Set up conditional breakpoint
-	WHERE	QUERY LOCATION	List current point of execution

Notes:

1. These commands are supported only with COBTEST FULL SCREEN debugging.
2. The DOWN, LEFT, RIGHT, and UP commands are executed by ISPF. They are not actual COBTEST commands.
3. HELP is supported only under TSO with COBTEST LINE MODE debugging.

4. LINK in COBTEST provides real storage for the data items in the LINKAGE SECTION. Debug Tool accomplishes this by declaring those data names. It then calls the COBOL program, passing the necessary parameters.
5. Debug Tool does not have a facility to list a range of names. However, it is possible to list all names for a block, or all names matching a specific pattern.
6. The PREVDISP command is no longer required. Debug Tool does not use ISPF for display.
7. The PRINTDD command is unnecessary for Debug Tool. The print file has a fixed ddname (although, the ddname can be specified at invocation.)
8. The RESTART command is available only with the programmable workstation. Note, the RESTART command does not retain break settings as it did in COBTEST.
9. The COBTEST SOURCE command is only supported in FULL SCREEN debugging. Also, the connotation is different from the TESTCOB SOURCE command.

Appendix F. Compiler option comparison

This appendix describes the Enterprise COBOL compiler options. It also shows how the Enterprise COBOL compiler options compare with those of VS COBOL II, OS/VS COBOL, and IBM COBOL. For complete descriptions of the Enterprise COBOL options, see the *Enterprise COBOL Programming Guide*.

Table 53. Compiler option comparison

	Available in				
Option	**OS/VS**	**VS II**	**IBM COBOL**	**Enterprise COBOL**	**Usage notes**
ADATA			X	X	Produces associated data file at compilation. NOADATA is the default. The Enterprise COBOL ADATA option replaces the COBOL/370 EVENTS option.
ANALYZE			X**		Causes the compiler to check the syntax of embedded SQL and CICS statements in addition to native COBOL statements.
ADV	X	X	X	X	Adds print control byte at beginning of records. ADV is the default.
ALOWCBL		X	X	X	Allows PROCESS or CBL statements in source program. You can only specified this option at installation time. ALOWCBL is the default.
APOST	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.
ARITH			X*	X	Sets the maximum number of digits that you can specify for decimal data and affects the precision of intermediate results. With ARITH(COMPAT) you can specify 18 digits in the PICTURE clause, fixed-point numeric literals, and arguments to NUMVAL and NUMVAL-C, and 28 digits in arguments to FACTORIAL. With ARITH(EXTEND) you can specify 31 digits in the PICTURE clause, fixed-point numeric literals, and arguments to NUMVAL and NUMVAL-C, and 29 digits in arguments to FACTORIAL.
AWO		X	X	X	Activates APPLY WRITE-ONLY processing for physical sequential files with VB format. NOAWO is the default.
BUF	X				Allocates buffer storage for compiler work data sets. In Enterprise COBOL, the BUFSIZE option replaces the OS/VS COBOL BUF option.

© Copyright IBM Corp. 1991, 2006

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
BUFSIZE		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.
CICS			X**	X	Enables the integrated CICS translator capability and specifies CICS options. NOCICS is the default.
CLIST	X				Produces a condensed PROCEDURE DIVISION listing plus tables and program statistics. NOCLIST is the default. The VS COBOL II , IBM COBOL, and Enterprise COBOL OFFSET option replaces the OS/VS COBOL CLIST option.
CMPR2		X	X		Specifies generation of IBM COBOL source code compatible with VS COBOL II, Release 2 or other VS COBOL II CMPR2 behavior. NOCMPR2 is the default. NOCMPR2 specifies the full use of all IBM COBOL language features (including language extensions for object-oriented COBOL and improved interoperability with C programs).
CODEPAGE				X	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.
COMPILE		X	X	X	Requests an unconditional full compilation. Other options are NOCOMPILE and NOCOMPILE(W|E|S). The default is NOCOMPILE(S). NOCOMPILE specifies unconditional syntax checking. NOCOMPILE(W|E|S) specify conditional syntax checking based on the severity of the error. COMPILE is equivalent to the OS/VS COBOL NOSYNTAX and NOCSYNTAX options. NOCOMPILE is equivalent to the OS/VS COBOL SYNTAX options. NOCOMPILE(W|E|S) is equivalent to the OS/VS COBOL CSYNTAX and SUPMAP options.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
COUNT	X				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. A similar function is provided with Debug Tool.
CURRENCY			X**	X	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. NOCURRENCY is the default and indicates that no alternate default currency sign is provided by the CURRENCY option.
DATA(24) DATA(31)		X	X	X	Specifies whether reentrant program data areas reside above or below the 16-MB line. With DATA(24) reentrant programs must reside below the 16-MB line. With DATA(31) reentrant programs can reside above the 16-MB line. DATA(31) is the default.
DATEPROC			X	X	Enables the millennium language extensions of the COBOL compiler. Options consist of DATEPROC(FLAG), DATEPROC(NOFLAG), DATEPROC(TRIG), DATEPROC(NOTRIG) and NODATEPROC.
DBCS		X	X	X	Tells the compiler to recognize DBCS shift-in and shift-out codes. DBCS is the default.
DBCSXREF=code		X	X	X	Specifies that an ordering program is to be used for cross-references to DBCS characters, where code sets parameters giving information about the DBCS Ordering Support Program. You can only specify DBCSXREF at installation time. DBCSXREF=NO is the default.
DECK	X	X	X	X	Generates object code as 80-character card images and places it in SYSPUNCH file. NODECK is the default.
DIAGTRUNC			X*	X	Causes the compiler to issue a severity-4 (warning) diagnostic for MOVE statements with numeric receivers when the receiving data has fewer integer positions than the sending data item or literal.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
DLL			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	Specifies that a system dump be produced at end of compilation. NODUMP is the default.
DYNAM	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)		X	X	X	Allows the compiler to accept user-supplied modules. (Each *string* is an optional user-supplied input string to the exit module, and each *mod* names a user-supplied exit module.) The ADT-id suboption is only available with COBOL for MVS & VM and COBOL for OS/390 & VM. NOEXIT is the default.
EXPORTALL			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.
FASTSRT		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	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		Specifies NOCMPR2 flagging for possible semantic changes from VS COBOL II, Release 2 or other programs with CMPR2 behavior. NOFLAGMIG is the default.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
FLAGSTD		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(SYM) option replaces the VS COBOL II FDUMP option.
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(ANSI) INTDATE(LILIAN)			X	X	Determines the starting date for integer format dates when used with date intrinsic functions. ANSI uses the COBOL 85 Standard starting date, where Day 1 = January 1, 1601. LILIAN uses the Language Environment Lilian starting date, where Day 1 = October 15, 1582. INTDATE(ANSI) is the default.
LANGUAGE		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. The default is NOLIB.
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	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 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
LIST		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	Produces a listing of the DATA DIVISION and, implicitly, declared items. NOMAP is the default. MAP replaces the OS/VS COBOL DMAP option.
MDECK				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	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	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.
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	Prints line numbers in error messages and listings. NONUMBER is the default. The NUMBER option replaces the OS/VS COBOL NUM option.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
NUMCLS		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: • *Enterprise COBOL Customization Guide*
NUMPROC		X	X	X	Handles packed/zoned decimal signs as follows: **NUMPROC(PFD)** Decimal fields assumed to have standard System/370 signs **NUMPROC(NOPFD)** The compiler does any necessary sign conversion of nonpreferred but valid signs. **NUMPROC(MIG)** Enterprise COBOL processes sign conversion in a manner very similar to OS/VS COBOL NUMPROC(NOPFD) is the default.
OBJECT		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.
OFFSET		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.
OPTIMIZE	X	X	X	X	Optimizes the object program. With IBM COBOL and Enterprise COBOL, OPTIMIZE has the suboptions of (STD/FULL). OPTIMIZE(FULL) provides improved runtime performance, over both the OS/VS COBOL and VS COBOL II OPTIMIZE option, because the compiler discards unused data items and does not generate code for any VALUE clauses for these data items. NOOPTIMIZE is the default.
OUTDD(SYSOUT) OUTDD(ddname)		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.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
PGMNAME			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 processed similarly to Release 1. PGMNAME(COMPAT) is the default.
PMAP	X				Produces a listing of assembler language expansion of source code. NOPMAP is the default. The VS COBOL II, IBM COBOL, and Enterprise COBOL LIST compiler option replaces the OS/VS COBOL PMAP option.
QUOTE	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.
RENT		X	X	X	Specifies reentrant code in object program. RENT is the default.
RMODE(AUTO) RMODE(24) RMODE(ANY)			X	X	Allows NORENT programs to have RMODE(ANY). RMODE(AUTO) is the default.
SEQ	X				Checks ascending sequencing of source statement line numbers. SEQ is the default. The VS COBOL II, IBM COBOL, and Enterprise COBOL SEQUENCE option replaces the OS/VS COBOL SEQ option.
SEQUENCE		X	X	X	Checks ascending sequencing of source statement line numbers. SEQUENCE is the default. SEQUENCE replaces the OS/VS COBOL SEQ option.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
SIZE(MAX) SIZE(nnnnn) SIZE(nnnK)		X	X	X	Specifies virtual storage to be used for compilation. SIZE(MAX) is the default.
SOURCE	X	X	X	X	Produces a listing of the source program and embedded messages. SOURCE is the default.
SPACE	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	Enables the DB2 coprocessor capability and specifies DB2 suboptions.
~ SQLSSCID ~ ~ ~ ~				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.
SSRANGE		X	X	X	At run time, checks validity of subscript, index, and reference modification references NOSSRANGE 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. NOSTATE is the default. The IBM COBOL and Enterprise COBOL TEST(NONE,SYM) option replaces the OS/VS COBOL STATE option.
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.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
~ 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 default is NOSXREF. 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. NOTERM is the default. The VS COBOL II, IBM COBOL, and Enterprise COBOL TERMINAL option replaces the OS/VS COBOL TERM option.
TERMINAL		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	Produces object code usable by Debug Tool for the product. NOTEST is the default. For details on the suboptions for the Enterprise COBOL TEST option, see the *Enterprise COBOL Programming Guide*.
THREAD				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	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. **TRUNC(STD)** Truncates numeric fields according to PICTURE specification of the binary receiving field **TRUNC(OPT)** Truncates numeric fields in the most optimal way **TRUNC(BIN)** Truncates binary fields based on the storage they occupy TRUNC(STD) is the default. For a complete description, see the *Enterprise COBOL Programming Guide*.

Table 53. Compiler option comparison (continued)

Option	Available in				Usage notes
	OS/VS	VS II	IBM COBOL	Enterprise COBOL	
TYPECHK			X		Enforces the rules for OO type conformance and issues diagnostics for any violations. The default is NOTYPECHK.
VBREF VBSUM	X	X	X	X	Produces a cross-reference listing of all verb types used in program. Only OS/VS COBOL supports VBSUM. The default is NOVBREF (as well as NOVBSUM for OS/VS COBOL).
WORD		X	X	X	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. NOWORD is the default.
XREF		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.
ZWB	X	X	X	X	Removes the sign from a signed numeric DISPLAY field when comparing it with an alphanumeric field. ZWB is the default.

Note:

- 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

Appendix G. Compiler limit comparison

The following table lists the compiler limits for Enterprise COBOL, 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	IBM COBOL	VS COBOL II	OS/VS COBOL
Size of program	999,999 lines	999,999 lines	999,999 lines	1-MB lines
Number of literals	4,194,303-B[1]	4,194,303-B[1]	4,194,303-B[1]	16-KB
Total length of literals	4,194,303-B[1]	4,194,303-B[1]	4,194,303-B[1]	32-KB 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	32,767-B	32,767-B	32,767-B	16-KB
IDENTIFICATION DIVISION				
ENVIRONMENT DIVISION				
CONFIGURATION SECTION				
SPECIAL-NAMES paragraph				
function-name IS	18	18	18	18
UPSI-n ... (switches)	0-7	0-7	0-7	0-7
alphabet-name IS ...	N/L	N/L	N/L	N/L
literal THRU/ALSO ...	256	256	256	256
INPUT-OUPUT Section				
FILE-CONTROL paragraph				
SELECT file-name ...	A maximum of 65,535 file names can be assigned external names	A maximum of 65,535 file names can be assigned external names	A maximum of 65,535 file names can be assigned external names	A maximum of 64-KB file names can be assigned external names
ASSIGN system-name ...	N/L	N/L	N/L	N/L
ALTERNATE RECORD KEY data-name ...	253	253	253	253
RECORD KEY length	N/L[2]	N/L[2]	N/L[2]	255
RESERVE integer (buffers)	255[3]	255[3]	255[3]	255[3]
I-O-CONTROL paragraph				

© Copyright IBM Corp. 1991, 2006

Compiler limit comparison

Language element	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
RERUN ON system-name ...	32,767	32,767	32,767	32-KB
integer RECORDS	16,777,215	16,777,215	16,777,215	16-MB
SAME RECORD AREA	255	255	255	255
FOR file-name ...	255	255	255	255
SAME SORT/MERGE AREA	N/L[4]	N/L[4]	N/L[4]	N/L[4]
MULTIPLE FILE ... file-name	N/L[4]	N/L[4]	N/L[4]	N/L[4]
DATA DIVISION				
FILE SECTION				
FD file-name ...	65,535	65,535	65,535	64-KB
LABEL data-name ... (if no optional clauses)	255	255	255	185
Label record length	80-B	80-B	80-B	80-B
DATA RECORD dnm ...	N/L[4]	N/L[4]	N/L[4]	N/L[4]
BLOCK CONTAINS integer	2,147,483,647[5]	2,147,483,647[5]	1,048,575[6]	32760
RECORD CONTAINS integer	1,048,575[6]	1,048,575[6]	1,048,575[6]	32-KB
Item length	1,048,575[6]	1,048,575[6]	1,048,575[6]	32-KB
SD file-name ...	65,535	65,535	65,535	64-KB
DATA RECORD dnm ...	N/L[4]	N/L[4]	N/L[4]	N/L[4]
Sort record length	32,751-B	32,751-B	32,751-B	32K-16-B
WORKING-STORAGE Section (items without the EXTERNAL attribute)	134,217,727-B	134,217,727-B	134,217,727-B	1-MB
WORKING-STORAGE Section (items with the EXTERNAL attribute)	134,217,727-B	134,217,727-B	134,217,727-B	1MB
77 data-names	134,217,727	16,777,215	16,777,215	1-MB
01-49 data-names	134,217,727	16,777,215	16,777,215	1-MB
88 condition-name ...	N/L	N/L	N/L	N/L
VALUE literal ...	N/L	N/L	N/L	N/L
66 RENAMES ...	N/L	N/L	N/L	N/L
PICTURE character-string	50	30	30	30
Numeric item digit positions	18 (or 31)[7]	18 (or 31)[7]	18	18
Num-edit character positions	249	249	249	127
PICTURE replication ( )	134,217,727	16,777,215	16,777,215	99999
PICTURE replication ( ), DBCS	67,108,863			
PICTURE replication ( ), national	67,108,863			
PIC repl (editing)	32,767	32,767	32,767	99999
DBCS Picture Reactivation	8,388,607	8,388,607		
Group item size:				
FILE SECTION	1,048,575	1,048,575		
Elementary item size	16,777,215	16,777,215	16,777,215	32-KB
VALUE initialization	134,217,727	16,777,215	16,777,215	64-KB
(Total length of value literals)	16,777,215	16,777,215	16,777,215	32-KB
OCCURS integer	134,217,727[1]	4,194,303[1]	4,194,303[1]	64-KB[1]
Total number of ODOs	7	7	7	3
Levels of OCCURS	16,777,215	16,777,215	16,777,215	32-KB
Table size	134,217,727	8,388,607	8,388,607	32-KB
Table element	134,217,727			
ASCENDING/DESCENDING KEY ...	12	12	12	12
(per OCCURS clause)	256B	256B	256B	256-B
Total length	12	12	12	12
INDEXED BY ... (index names)	65,535	65,535	65,535	64-KB
Total num of indexes (index names)	32,765	32,765	32,765	32-KB
Size of relative index				
Linkage Section	134,217,727	134,217,727	134,217,727	1-MB

Language element	Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
Total 01 + 77 (data items)	N/L	N/L	N/L	255
PROCEDURE DIVISION				
Procedure + constant area	4,194,303[1]	4,194,303[1]	4,194,303[1]	1M+32-KB
USING identifier ...	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 identifier ...	N/L	N/L	N/L	N/L
ALTER pn1 TO pn2 ...	4,194,303[1]	4,194,303[1]	4,194,303[1]	64-KB
CALL ... BY CONTENT id	2,147,483,647	2,147,483,647	2,147,483,647	N/L
CALL literal ...	4,194,303[1]	4,194,303[1]	4,194,303[1]	N/L
USING id/lit ...	16,380	16,380	16,380	N/L
Active programs in run unit	32,767	32,767	32,767	32-KB
RES/DYN number of names called	N/L	N/L	N/L	64-KB
CANCEL id/lit ...	N/L	N/L	N/L	N/L
CLOSE file-name ...	N/L	N/L	N/L	N/L
COMPUTE identifier ...	N/L	N/L	N/L	N/L
DISPLAY id/lit ...	N/L[8]	N/L[8]	N/L[8]	N/L[8]
DIVIDE identifier ...	N/L	N/L	N/L	N/L
ENTRY USING id/lit ...	N/L	N/L	N/L	N/L
EVALUATE ... subjects	64	64	64	N/L
EVALUATE ... WHEN clauses	256	256	256	N/L
GO pn ... DEPENDING	255	255	255	2031
INSPECT TALLYING/REPLACING	N/L	N/L	N/L	15
MERGE file-name				
ASCENDING/DESCENDING KEY ...	N/L	N/L	N/L	12
Total key length	4092-B[9]	4092-B[9]	4092-B[9]	256
USING file-name ...	16[10]	16[10]	16[10]	16[10]
MOVE id/lit TO id ...	N/L	N/L	N/L	N/L
MULTIPLY identifier ...	N/L	N/L	N/L	N/L
OPEN file-name	N/L	N/L	N/L	N/L
PERFORM	4,194,303	4,194,303	4,194,303	64-KB
SEARCH ... WHEN ...	N/L	N/L	N/L	N/L
SEARCH ALL ... WHEN ...	12	12	12	12
SET index/id ... TO	N/L	N/L	N/L	N/L
SET index ... UP/DOWN	N/L	N/L	N/L	N/L
SORT file-name				
ASCENDING/DESCENDING KEY	N/L	N/L	N/L	12
Total key length	4092-B[9]	4092-B[9]	4092-B[9]	256
USING file-name ...	16[10]	16[10]	16[10]	16[10]
STRING identifier ...	N/L	N/L	N/L	N/L
DELIMITED id/lit ...	N/L	N/L	N/L	N/L
UNSTRING id				
DELIMITED id/lit OR id/lit ...	255	255	255	15
INTO id/lit ...	N/L	N/L	N/L	N/L
USE ... ON file-name ...	N/L	N/L	N/L	N/L

Notes:

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. Treated as comment; there is no limit.
5. Requires LBI (Large Block Interface) support provided by OS/390 DFSMS, Version 2 Release 10.0, or z/OS. On OS/390 systems with earlier releases of DFSMS, the limit is 32,767 bytes.

6. The compiler limit is shown, but QSAM limits it to 32,767 bytes.
7. 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.
8. The compiler limit is shown; however, Language Environment limits the maximum length of a data item you can display with DISPLAY UPON SYSOUT to 16,384.
9. The limit is 4088 bytes if EQUALS is coded on the OPTION control statement.
10. SORT limit.

For additional information on using DISPLAY with OS/VS COBOL programs, see "Understanding SYSOUT output changes" on page 70.

For additional information on using DISPLAY with VS COBOL II programs, see "Understanding SYSOUT output changes" on page 90.

Appendix H. Preventing file status 39 for QSAM files

This appendix provides guidelines to help prevent common file status 39 problems for QSAM files, which are due to mismatches in the attributes for file organization, record format (fixed or variable), record length, or the code set.

Processing existing files

When your program processes an existing file, code the description of the file in your COBOL program to be consistent with the file attributes of the data set, for example:

Format-V files or Format-S files	The maximum record length specified in your program must be exactly 4 bytes smaller than the length attribute of the data set.
For Format-F files	The record length specified in your program must exactly match the length attribute of the data set.
For Format-U files	The maximum record length specified in your program must exactly match the length attribute of the data set.

Note: Remember that information in the JCL overrides information in the data set label.

For details on how record lengths are determined from the FD entry and record descriptions in your program, see *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).
```

© Copyright IBM Corp. 1991, 2006

Defining fixed-length records

To define fixed-length records in your program, use either the RECORD CONTAINS integer clause, or omit this clause and specify all level-01 record descriptions to be the same fixed size. In either case, use a value that equals the value of the length attribute of the data set. When you intend to use the same program to process different files at execution and the files have differing fixed-length record lengths, the recommended way to avoid record-length conflicts is to code RECORD CONTAINS 0.

If the existing file is an ASCII data set (DCB=(OPTCD=Q)), you must specify the CODE-SET clause in the program's FD entry for the file.

Converting existing files that do not match the COBOL record

You can re-allocate a new file with the matching LRECL, copy the data from an existing file to the new file, then use the new file as input.

Processing new files

When your COBOL program will write records to a new file which is made available before the program is run, ensure that the file attributes you specify in the DD statement or the allocation do not conflict with the attributes you have specified in your program. In most cases, you only need to specify a minimum of parameters when predefining your files, as illustrated in the following example of a DD statement related to the FILE-CONTROL and FD entries in your program:

```
JCL DD Statement:

 1
//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 9. 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
      RECORDING MODE IS V
      RECORD CONTAINS 10 TO 50 CHARACTERS.
  01  COMMUTER-RECORD-A           PIC X(10).
  01  COMMUTER-RECORD-B           PIC X(50).
```

The LRECL in your DD statement or allocation should be 54.

Processing files dynamically created by COBOL

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 I. Overriding linkage editor defaults

This appendix gives you the instructions necessary to override the default AMODE and RMODE settings assigned by the Enterprise COBOL compiler based on the use of the RENT and RMODE compiler options.

When to override the default settings

For load modules with both Enterprise COBOL and OS/VS COBOL programs, you must override the default AMODE setting to AMODE(24) when the load module contains a Enterprise COBOL program compiled with NORENT. (For programs compiled with RENT, no action is necessary. The linkage editor automatically assigns the correct AMODE setting.)

Exception: Do not override the default AMODE/RMODE settings in the following cases:

AMODE
For load modules that contain VS COBOL II programs compiled NORES or any OS/VS COBOL programs, do not specify a linkage editor override of AMODE ANY or AMODE 31. The only exception is if the programs are external entry points called by the system or through system services and the logic of the application can guarantee, through appropriate AMODE switching, that these programs will be entered in AMODE 24. These programs will not switch AMODEs when they statically call other programs.

RMODE
For load modules that contain VS COBOL II programs compiled NORES and NORENT or any OS/VS COBOL programs, do not specify a linkage editor override of RMODE ANY. This is because certain control blocks contained in the object modules produced by the compiler must reside below the 16-MB line.

How to override the defaults

To override the defaults, specify AMODE or RMODE with one of the following:

- EXEC statement of your link-edit job step, where *programname* refers to the linkage editor or program binder, such as IEWL or HEWL:

  ```
  //LKED    EXEC    PGM=programname,
  //            PARM='AMODE=xx,RMODE=yy'
  ```

- The linkage editor mode control statements

  ```
  MODE AMODE(xx),RMODE(yy)
  ```

- One of the following TSO commands LINK or LOADGO

  LINK(*dsn-list*) AMODE(*xx*) RMODE(*yy*)

  LOADGO(*dsn-list*) AMODE(*xx*) RMODE(*yy*)

See your *Linkage Editor and Loader User's Guide* for allowable *xx* and *yy* and linkage editor mode control statements.

© Copyright IBM Corp. 1991, 2006

Overriding linkage editor defaults

The linkage editor uses a program's AMODE attribute to determine whether a program invoked using ATTACH, LINK, XCTL, or LOAD/BASSM is to receive control in 24-bit or 31-bit addressing mode. The loader uses the RMODE attribute to determine whether a program must be loaded into virtual storage below 16-MB, or can reside anywhere in virtual storage (above or below 16-MB).

Appendix J. Link-edit example

This appendix provides an example of JCL that shows how to replace the current library routines in a load module with the Language Environment library routines. The SCEESAMP data set contains 3 sample jobs (IGZWRLKA, IGZWRLKB, and IGZRLKC) to assist in relink-editing OS/VS COBOL or VS COBOL II load modules.

```
//*********************************************************************
//*                                                                   *
//*  RELINK A LOAD MODULE THAT HAS BOTH OS/VS COBOL PROGRAMS          *
//*  AND VS COBOL II PROGRAMS WITH Language Environment.              *
//*                                                                   *
//*********************************************************************
//*
//LINK   EXEC  PGM=IEWL,PARM='LIST,MAP,XREF'
//SYSPRINT DD  SYSOUT=*
//*********************************************************************
//*   CHANGE 'ZZZZZZ.SCEELKED' IN THE FOLLOWING DD STATEMENT TO       *
//*   THE Language Environment SCEELKED DATA SET NAME.                *
//*                                                                   *
//*   CHANGE 'XXXXXX' IN THE FOLLOWING DD STATEMENT TO                *
//*   THE DATA SET NAME WHICH CONTAINS THE LOAD MODULE.               *
//*                                                                   *
//*   CHANGE 'YYYYYY' IN THE FOLLOWING DD STATEMENT TO                *
//*   THE DATA SET NAME WHICH THE RELINK-EDITED LOAD                  *
//*   MODULE SHOULD BE SAVED INTO.                                    *
//*                                                                   *
//*********************************************************************
//SYSLIB   DD  DSN=ZZZZZZ.SCEELKED,DISP=SHR
//LOADLIB  DD  DSN=XXXXXX,DISP=SHR
//SYSLMOD  DD  DSN=YYYYYY,DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(2,2)),DISP=NEW
//*********************************************************************
//*   CHANGE 'UUUUUU' IN THE FOLLOWING INCLUDE STATEMENT              *
//*   TO THE LOAD MODULE NAME.                                        *
//*                                                                   *
//*   CHANGE 'VVVVVV' IN THE FOLLOWING NAME STATEMENT TO              *
//*   THE RELINK-EDITED LOAD MODULE NAME.                             *
//*                                                                   *
//*   CHANGE 'EEEEEE' IN THE FOLLOWING ENTRY STATEMENT TO             *
//*   THE RELINK-EDITED LOAD MODULE ENTRY POINT NAME OR               *
//*   OMIT THE ENTRY STATEMENT IF IT IS NOT REQUIRED.                 *
//*                                                                   *
//*********************************************************************
//SYSLIN   DD  *
  REPLACE ILBOxxxx  1
    .
    .
    .
  REPLACE IGZxxxx  2
    .
    .
    .
  INCLUDE LOADLIB(UUUUUU)
  ENTRY   EEEEEE
  NAME    VVVVVV(R)
/*
```

Where 1 and 2 represent the following:

© Copyright IBM Corp. 1991, 2006

1

```
REPLACE ILBOABN
REPLACE ILBOACP
REPLACE ILBOACS
REPLACE ILBOANE
REPLACE ILBOANF
REPLACE ILBOATB
REPLACE ILBOBEG
REPLACE ILBOBID
REPLACE ILBOBIE
REPLACE ILBOBII
REPLACE ILBOBUG
REPLACE ILBOCHN
REPLACE ILBOCJS
REPLACE ILBOCKP
REPLACE ILBOCLS
REPLACE ILBOCMM
REPLACE ILBOCOM0
REPLACE ILBOCT1
REPLACE ILBOCVB
REPLACE ILBODBE
REPLACE ILBODBG
REPLACE ILBODCI
REPLACE ILBODSP
REPLACE ILBODSS
REPLACE ILBODTE
REPLACE ILBOD01
REPLACE ILBOD10
REPLACE ILBOD11
REPLACE ILBOD12
REPLACE ILBOD13
REPLACE ILBOD14
REPLACE ILBOD20
REPLACE ILBOD21
REPLACE ILBOD22
REPLACE ILBOD23
REPLACE ILBOD24
REPLACE ILBOD25
REPLACE ILBOD26
REPLACE ILBOEFL
REPLACE ILBOERR
REPLACE ILBOETB
REPLACE ILBOEXT
REPLACE ILBOFLW
REPLACE ILBOFPW
REPLACE ILBOGDO
REPLACE ILBOGPW
REPLACE ILBOIDB
REPLACE ILBOIDR
REPLACE ILBOIDT
REPLACE ILBOIFB
REPLACE ILBOIFD
REPLACE ILBOINS
REPLACE ILBOINT
REPLACE ILBOITB
REPLACE ILBOIVL
REPLACE ILBOLBL
REPLACE ILBOMRG
REPLACE ILBOMSG
REPLACE ILBOMSC
REPLACE ILBONBL
REPLACE ILBONED
REPLACE ILBONTR
REPLACE ILBOOCR
REPLACE ILBOPRM
REPLACE ILBOPTV
REPLACE ILBOQIO
REPLACE ILBOQSS
REPLACE ILBOQSU
REPLACE ILBOREC
REPLACE ILBORNT
REPLACE ILBOSAM
REPLACE ILBOSCD
REPLACE ILBOSCH
REPLACE ILBOSDB
REPLACE ILBOSGM
REPLACE ILBOSMG
REPLACE ILBOSMV
REPLACE ILBOSND
REPLACE ILBOSNT
REPLACE ILBOSPI
REPLACE ILBOSPN
REPLACE ILBOSPA
REPLACE ILBOSQA
REPLACE ILBOSRT
REPLACE ILBOSRV
REPLACE ILBOSSN
REPLACE ILBOSTG
REPLACE ILBOSTI
REPLACE ILBOSTN
REPLACE ILBOSTR
REPLACE ILBOSTT
REPLACE ILBOSYN
REPLACE ILBOTC0
REPLACE ILBOTC2
REPLACE ILBOTC3
REPLACE ILBOTEF
REPLACE ILBOTRN
REPLACE ILBOUST
REPLACE ILBOUTB
REPLACE ILBOVCO
REPLACE ILBOVIO
REPLACE ILBOVMO
REPLACE ILBOVOC
REPLACE ILBOVTR
REPLACE ILBOWAT
REPLACE ILBOWTB
REPLACE ILBOXDI
REPLACE ILBOXMU
REPLACE ILBOXPR
```

2

```
REPLACE IGZCA2D
REPLACE IGZCACP
REPLACE IGZCACS
REPLACE IGZCANE
REPLACE IGZCANF
REPLACE IGZCBID
REPLACE IGZCBUG
REPLACE IGZCCCO
REPLACE IGZCCLS
REPLACE IGZCCTL
REPLACE IGZCCVB
REPLACE IGZCD2A
REPLACE IGZCDIF
REPLACE IGZCDSP
REPLACE IGZCFDP
REPLACE IGZCFDW
REPLACE IGZCFPW
REPLACE IGZCGDR
REPLACE IGZCIDB
REPLACE IGZCINS
REPLACE IGZCIN1
REPLACE IGZCIN2
REPLACE IGZCIPS
REPLACE IGZCIVL
REPLACE IGZCKCL
REPLACE IGZCLNK
REPLACE IGZCMSF
REPLACE IGZCMST
REPLACE IGZCONV
REPLACE IGZCONVX
REPLACE IGZCSCH
REPLACE IGZCSMV
REPLACE IGZCSPA
REPLACE IGZCSPC
REPLACE IGZCSSN
REPLACE IGZCSTG
REPLACE IGZCTCO
REPLACE IGZCULE
REPLACE IGZCUST
REPLACE IGZCVIN
REPLACE IGZCVMO
REPLACE IGZCXDI
REPLACE IGZCXFR
REPLACE IGZCXMU
REPLACE IGZCXPR
REPLACE IGZEABX
REPLACE IGZEABN
REPLACE IGZEBRG
REPLACE IGZEBST
REPLACE IGZECKP
REPLACE IGZECMS
REPLACE IGZEDBR
REPLACE IGZEDBW
REPLACE IGZEDTE
REPLACE IGZEINP
REPLACE IGZEMSG
REPLACE IGZENRI
REPLACE IGZENRT
REPLACE IGZEOPD
REPLACE IGZEOPT
REPLACE IGZEOUT
REPLACE IGZEPRM
REPLACE IGZEPTV
REPLACE IGZEQBL
REPLACE IGZEQOC
REPLACE IGZERRE
REPLACE IGZESAT
REPLACE IGZESMG
REPLACE IGZESNP
REPLACE IGZESPM
REPLACE IGZESTA
REPLACE IGZETRM
REPLACE IGZETUN
REPLACE IGZEVAM
REPLACE IGZEVEX
REPLACE IGZEVIO
REPLACE IGZEVOC
REPLACE IGZEVOP
REPLACE IGZEVSV
REPLACE IGZTCAM2
REPLACE IGZTCAM4
REPLACE IGZTCM21
REPLACE IGZTCM41
REPLACE IGZTCM42
```

Link-edit example

Appendix K. DB2 coprocessor integration

A coprocessor approach eliminates the need for precompilation with the DB2 precompiler in COBOL programs containing SQL statements.

The coprocessor approach 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.

A separate precompiler approach is still supported in 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 coprocessor approach requires DB2, version 7 or later. The benefits of a coprocessor approach include:

- 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.
- The need for an intermediate data set to hold the translated but not compiled version of the source program is eliminated.
- There is only one output listing instead of two.
- Nested programs that contain EXEC SQL statements can be held in separate files and included through a COPY statement.
- REPLACE statements can now affect EXEC SQL statements.

The following job stream shows an example of using the DB2 precompiler:

```
//DB2PRE JOB ...,
//NOTIFY=JDOE,MSGCLASS=A,CLASS=A,TIME=(0,5),
//REGION=10M,MSGLEVEL=(1,1)MIN00030
//PC       EXEC PGM=DSNHPC,
//         PARM='HOST(COB2),QUOTE,APOSTSQL,SOURCE,XREF'
//DBRMLIB DD DSN=JDOE.DBRMLIB.DATA(COBTEST),DISP=SHR
//STEPLIB  DD DSN=DSN710.SDSNLOAD,DISP=SHR
//SYSCIN   DD DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
//            SPACE=(800,(500,500))
//SYSPRINT DD SYSOUT=*
//SYSTERM  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.COBTEST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 RES PIC X(10).
EXEC SQL
```

© Copyright IBM Corp. 1991, 2006

```
INCLUDE SQLCA
END-EXEC.
PROCEDURE DIVISION.
EXEC SQL
SELECT COL1 INTO :RES FROM TABLE1
END-EXEC.

//COB EXEC PGM=IGYCRCTL,
//PARM=(QUOTE,NODYNAM,ADV,'BUF(12288)',SOURCE,NOXREF)
//STEPLIB DD DSN=IGY.V3R2M0.SIGYCOMP,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)
```

The following example shows the integrated SQL coprocessor:

```
//DB2INT JOB (JDOE,E264,090,D48),'John Doe',
//NOTIFY=JDOE,MSGCLASS=A,CLASS=A,TIME=(0,5),
//REGION=10M,MSGLEVEL=(1,1)
//COB EXEC PGM=IGYCRCTL,
//PARM=(QUOTE,NODYNAM,ADV,'BUF(12288)',SOURCE,NOXREF)
//STEPLIB DD DSN=IGY.V3R2M0.SIGYCOMP,DISP=SHR
//                    DD DSN=DSN710.SDSNLOAD,DISP=SHR
//DBRMLIB DD DSN=JDOE.DBRMLIB.DATA(COBTEST),DISP=SHR
//SYSIN DD *
CBL SQL('HOST(COB2),QUOTE,APOSTSQL,SOURCE,XREF'),LIB
IDENTIFICATION DIVISION.
PROGRAM-ID.COBTEST.
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.
//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)
```

Considerations when moving to the coprocessor

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.

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.

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.

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.

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.

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.

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.

Appendix L. IMS considerations

When running any combination of OS/VS COBOL, VS COBOL II, IBM COBOL, and Enterprise COBOL programs under Language Environment, you need to know:

- Unsupported features from VS COBOL II
- Compiler options relevant for programs run on IMS
- Compiling and linking COBOL programs for running under IMS
- ENDJOB/NOENDJOB compiler option requirements
- Recommended modules for preload
- Condition handling using CBLTDI on IMS
- Performance consideration when running OS/VS COBOL programs
- The use of a GTF trace to determine which modules are loaded
- DFSPCC20 modification is unsupported

Unsupported VS COBOL II features

BLDL user exit unsupported

The VS COBOL II BLDL user exit is not supported in Language Environment. No replacement is available. However, Language Environment does have a load notification exit that might meet your needs. For information on the load notification exit, see *Language Environment for z/OS Customization*.

LIBKEEP unsupported

The VS COBOL II LIBKEEP runtime option is not supported in Language Environment. For alternatives to LIBKEEP, see "Existing applications using LIBKEEP" on page 94.

Compiler options relevant for programs run on IMS

Table 54 lists Enterprise COBOL compiler options relevant to COBOL IMS applications.

Table 54. Compiler options relevant for Enterprise COBOL programs run on IMS

Compiler option	Comments
RENT	The RENT compiler option causes Enterprise COBOL to produce reentrant code and allows you to place the COBOL modules in the LPA (Link Pack Area) or ELPA (Extended Link Pack Area) and thus shared among dependent regions under IMS. Also, the modules cannot be overwritten, since the LPA/ELPA have a storage protect key. RENT allows you to run your IMS programs in either preload or nonpreload mode, without compiling with different options.

Compiling and linking COBOL programs 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.

© Copyright IBM Corp. 1991, 2006

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, VS COBOL II, and OS/VS COBOL programs, the following compiler options are recommended:

Table 55. Recommended compiler options for applications with mixed COBOL programs

Enterprise COBOL	IBM COBOL	VS COBOL II	OS/VS COBOL
RENT	RENT	RENT and RES	RES and NOENDJOB for preloaded programs RES and ENDJOB for nonpreloaded programs

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 either RENT or NORENT RMODE(ANY), depending on your IMS environment.

With IMS, the data for IMS application programs can reside above the 16-MB line, and you can use DATA(31) RENT, or RMODE(ANY) NORENT 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.

ENDJOB/NOENDJOB compiler option requirements

If you are running a mixture of OS/VS COBOL and Enterprise COBOL programs, at least one of the OS/VS COBOL programs must have been compiled with the ENDJOB option if the programs are not preloaded.

Specifying the ENDJOB option allows the COBOL application and the Language Environment runtime environment to be completely cleaned up at program termination.

Note: If LRR is used at the same time as OS/VS COBOL and Enterprise COBOL, the runtime environment will not be completely cleaned up, because LRR forces NOENDJOB behavior.

Preloading requirements

If you preload ILBOCOM, you must also preload the following library routines:
- ILBOCMM, ILBONTR, and ILBOSRV
- ILBOACS, ILBOCVB, and ILBOINS if an OS/VS COBOL RES program uses the INSPECT and/or UNSTRING verbs.

Last used state behavior under Language Environment

The OS/VS COBOL NOENDJOB/ENDJOB compiler option produces different results when running with the Language Environment library than when running with the OS/VS COBOL library. When specifying NOENDJOB and running under OS/VS COBOL, your subprograms are always entered in the last used state.

In the Language Environment runtime environment, when nonpreloaded OS/VS COBOL subprograms that are compiled with RES and NOENDJOB terminate, the programs and their internal work areas are deleted; their external work areas (storage acquired with a GETMAIN) and library routines remain in dynamic storage. When ENDJOB is used instead of NOENDJOB, there is essentially no difference except that the external work areas and library routines are deleted as well.

Because Enterprise COBOL programs and their resources (all work areas and library routines) are always deleted at termination, the way ENDJOB and NOENDJOB work in the Language Environment runtime environment allows both Enterprise COBOL and OS/VS COBOL programs to be treated similarly with respect to last used state.

In the OS/VS COBOL runtime environment, ENDJOB was the strongly recommended compiler option when executing IMS transactions invoking nonpreloaded COBOL programs.

In the VS COBOL II runtime environment when using the LIBKEEP runtime option, OS/VS COBOL programs compiled with ENDJOB were treated as NOENDJOB. NOENDJOB was required for preloaded OS/VS COBOL programs.

Note: When using LRR, NOENDJOB behavior is always in effect.

When programs remain in the last-used state

A program remains in last-used state between COBOL transactions only when all the following conditions are true:
- It is an OS/VS COBOL program compiled with the NOENDJOB option or LRR is being used.
- It has been link-edited with the REUS option.
- It has been preloaded.
- It is a dynamically called subprogram or a statically called subprogram that has, itself, been called by a dynamically called subprogram.

Recommended modules for preload

IMS allows application programs to be preloaded (to remain resident in storage after each use). Preloading can improve performance; if the program is already resident in storage, requests for the program can be handled faster. No time is wasted bringing it into storage from an external medium.

Enterprise COBOL programs

For a list of recommended modules to preload, see the *IBM Enterprise COBOL Version 3 Release 1 Performance Tuning Paper* on the Web. Go to the Library Section, at `www.ibm.com/software/awdtools/cobol/library`.

OS/VS COBOL programs

Table 56 shows a sample list of OS/VS COBOL compatibility subroutines to preload if you use the OS/VS COBOL RES compiler option. If you run OS/VS COBOL programs with Language Environment, you might also want to preload the modules listed in the *Performance Paper*.

Table 56. Sample list for preloaded ILBO modules

ILBOCHN0	ILBOCMM0	ILBOCOM0
ILBOCVB0	ILBODTE0	ILBOETB0
ILBOGDO0	ILBOINS0	ILBOITB0
ILBONTR0	ILBOSCH0	ILBOUST0
ILBOWTB0	ILBOSRV0	ILBOSTG0
ILBOSTT2	ILBOTRN0	

When running under Language Environment:

- Continue to preload the same ILBO library routines as you preloaded when using the OS/VS COBOL library. (These ILBO routines are included in Language Environment.)
- Preload ILBOSTT2 (and put it in the preload list twice) *if* you preloaded ILBOSTT for your OS/VS COBOL programs running with the OS/VS COBOL library.

Condition handling using CBLTDLI on IMS

1. DO NOT handle the condition with condition handling. If you do attempt to handle the errors using condition handlers, the integrity of your IMS database is at risk.
2. Use the ABTERMENC(ABEND) and TRAP(ON) runtime options to ensure that the application terminates abnormally and transforms all abnormal terminations into operating system abends to cause IMS roll backs.

Performance consideration when running OS/VS COBOL programs

If most of the programs scheduled in your IMS region are OS/VS COBOL programs, a performance benefit might be achieved by specifying lower values for the Language Environment storage related runtime options. For example, you could use the following storage values:

```
STACK(16K)
HEAP(4K,4K,ANY,4K,4K)
BELOWHEAP(4K)
ANYHEAP(4K)
```

Using a GTF trace to determine which modules are loaded

Some customers collect GTF trace information and look at the SVC 8 (LOAD) trace information to determine which application programs are loaded. Language Environment uses both SVC 8 and SVC 122 to load programs.

DFSPCC20 modification unsupported

OS/VS COBOL allowed the use of the ENDJOB option for both preloaded and nonpreloaded programs with a user modification to the IMS Program Controller (DFSPCC20).

The user modification included code that would store the addresses of certain preloaded subroutines and might cause intermittent problems at execution. This user modification (or any user supplied code which performs the same task) must be removed before running under Language Environment.

Appendix M. TSO considerations

This appendix describes conversion considerations for programs running on TSO. It includes information on using REXX execs.

Using REXX execs

When you run a COBOL program from a REXX exec, you need to be aware of the differences in the parameter list formats for using the different "address" options. When 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.

© Copyright IBM Corp. 1991, 2006

Appendix N. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
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.

Programming interface information

This book is intended to help you write programs using IBM Enterprise COBOL for z/OS. This Compiler and Runtime 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.

© Copyright IBM Corp. 1991, 2006

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
C/370
CICS
CICS/ESA
COBOL/370
DB2
DFSORT
IBM
IMS
Language Environment
MVS
OS/2
OS/390
S/390
SAA
SOM
SOMobjects
System Object Model
System/370
VisualAge
VSE/ESA
WebSphere
z/OS
zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Unicode™ is a trademark of the Unicode® Consortium.

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.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

List of resources

IBM Enterprise COBOL for z/OS

Compiler and Runtime Migration Guide, GC27-1409
Customization Guide, GC27-1410
Debug Tool User's Guide, SC18-9012
Debug Tool Reference Manual and Messages, SC18-9010
Fact Sheet, GC27-1407
Language Reference, SC27-1408
Licensed Program Specifications, GC27-1411
Programming Guide, SC27-1412

Language Environment for z/OS

Concepts Guide, SA22-7567
Debugging Guide, GA22-7560
Run-Time Messages, SA22-7566
Customization, SA22-7564
Programming Guide, SA22-7561
Programming Reference, SA22-7562
Run-Time Migration Guide, GA22-7565
Writing Interlanguage Applications, SA22-7563

Language Environment for OS/390

Concepts Guide, GC28-1945
Debugging Guide and Run-Time Messages SC28-1942
Customization, SC28-1941
Programming Guide, SC28-1939
Programming Reference, SC28-1940
Run-Time Migration Guide, SC28-1944
Writing Interlanguage Applications, SC28-1943

Related publications

- **IBM C/370**
  Migration Guide, SC09-1359
  Programming Guide, SC09-1841
  Language Reference, SC09-1762
  General Information, GC09-1358
  Diagnosis Guide, LY09-1806
  Licensed Program Specifications, GC09-1357
  Reference Summary, SX09-1247
- **IBM PL/I for MVS & VM**
  Fact Sheet, GC26-3112

© Copyright IBM Corp. 1991, 2006

Licensed Program Specification, GC26-3116

Compiler and Runtime Migration Guide, SC26-3118

Installation and Customization under MVS, SC26-3119

Installation and Customization under CMS, SC26-3120

Programming Guide, SC26-3113

Language Reference, SC26-3114

Reference Summary, SX26-3821

Diagnosis Guide, SC26-3149

Messages and Codes, SC26-3229

- **DB2**

  Application Programming and SQL Guide, SC26-9933

- **CCCA**

  COBOL and CICS/VS Command Level Conversion Aid Program Description and Operations Manual, SB11-6432

- **CICS/ESA**

  Application Programming Guide, SC33-1169

  Application Programming Reference, SC33-1170

  Customization Guide, SC33-1165

  External CICS Interface, SC33-1390

  Sample Applications Guide, SC33-1173

- **COBOL Report Writer**

  COBOL Report Writer Precompiler Programmer's Manual, SC26-4301

  COBOL Report Writer Precompiler Installation and Operation for MVS and CMS, SC26-4302

- **IMS**

  Application Programming: EXEC DLI Commands for CICS and IMS , SC27-1288

- z/OS TSO/E and ISPF
  - *TSO/E Primer*, SA22-7787
  - *TSO/E User's Guide*, SA22-7794
  - *ISPF User's Guide Volume I*, SC34-4822

Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms may or may not have the same meaning in other languages.

This glossary includes terms and definitions from the following publications:

- American National Standard *ANSI INCITS 23-1985, Programming languages - COBOL* as amended by *ANSI INCITS 23a-1989, Programming Languages - COBOL - Intrinsic Function Module for COBOL*, and *ANSI INCITS 23b-1993, Programming Languages - Correction Amendment for COBOL*
- *ANSI X3.172-1990, American National Standard Dictionary for Information Systems*

Copies can be purchased from the American National Standards Institute, 25 West 43rd Street, New York, New York 10036.

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.

*** access mode.** The manner in which records are to be operated upon within a file.

*** actual decimal point.** The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

*** alphabet-name.** A user-defined word, in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, that assigns a name to a specific character set and/or collating sequence.

*** alphabetic character.** A letter or a space character.

*** alphanumeric character.** Any character in the computer's character set.

alphanumeric-edited character. A character within an alphanumeric character-string that contains at least one B, 0 (zero), or / (slash).

*** alphanumeric function.** A function whose value is composed of a string of one or more characters from the computer's character set.

*** alternate record key.** A key, other than the prime record key, whose contents identify a record within an indexed file.

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.

© Copyright IBM Corp. 1991, 2006

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

big-endian. Default format used by the mainframe and the AIX workstation to store binary data. In this format, the least significant digit is on the highest address. Compare with "little-endian".

binary item. A numeric data item represented in binary notation (on the base 2 numbering system). Binary items have a decimal equivalent consisting of the decimal digits 0 through 9, plus an operational sign. The leftmost bit of the item is the operational sign.

binary search. A dichotomizing search in which, at each step of the search, the set of data elements is divided by two; some appropriate action is taken in the case of an odd number.

*** block.** A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

breakpoint. A place in a 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 representing a SOM class.

* **clause.** An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

CMS (Conversational Monitor System). A virtual machine operating system that provides general interactive, time-sharing, problem solving, and program development capabilities, and that operates only under the control of the VM/SP control program.

* **COBOL character set.** The complete COBOL character set consists of the characters listed below:

Character	Meaning
0,1...,9	digit
A,B,...,Z	uppercase letter
a,b,...,z	lowercase letter
␣	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 Data Division, and that bear a definite hierarchic relationship to each other.

copybook. A file or library member containing a sequence of code that is included in the source program at compile time using the COPY statement. The file can be created by the user, supplied by COBOL, or supplied by another product.

CORBA. The Common Object Request Broker Architecture established by the Object Management Group. IBM's *Interface Definition Language* used to describe the *interface* for SOM classes is fully compliant with CORBA standards.

* **counter.** A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

cross-reference listing. The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

currency sign. The character '$' of the COBOL character set or that character defined by the CURRENCY compiler option. If the NOCURRENCY compiler option is in effect, the currency sign is defined as the character '$'.

currency symbol. The character defined by the CURRENCY compiler option or by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If the NOCURRENCY compiler option is in effect for a COBOL source program and the CURRENCY SIGN clause is also **not** present in the source program, the currency symbol is identical to the currency sign.

* **current record.** In file processing, the record that is available in the record area associated with a file.

* **current volume pointer.** A conceptual entity that points to the current volume of a sequential file.

D

* **data clause.** A clause, appearing in a data description entry in the DATA DIVISION of a COBOL program, that provides information describing a particular attribute of a data item.

* **data description entry .** An entry in the DATA DIVISION of a COBOL program that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

DATA DIVISION. 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.

do-until. In structured programming, a do-until loop will be executed at least once, and until a given condition is true. In COBOL, a TEST AFTER phrase used with the PERFORM statement functions in the same way.

do-while. In structured programming, a do-while loop will be executed if, and while, a given condition is true. In COBOL, a TEST BEFORE phrase used with the PERFORM statement functions in the same way.

Double-Byte Character Set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require Double-Byte Character Sets. Because each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

*** dynamic access.** An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement.

dynamic link library. A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

Dynamic Storage Area (DSA). Dynamically acquired storage composed of a register save area and an area available for dynamic storage allocation (such as program variables). DSAs are generally allocated within STACK segments managed by Language Environment.

E

*** EBCDIC (Extended Binary-Coded Decimal Interchange Code).** A coded character set consisting of 8-bit coded characters.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set.

edited data item. A data item that has been modified by suppressing zeroes and/or inserting editing characters.

*** editing character.** A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
b	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign

, comma (decimal point)
. period (decimal point)
/ slant (virgule, slash)

element (text element). One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type.

* **elementary item.** A data item that is described as not being further logically subdivided.

enclave. 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 header.** A combination of words, followed by a separator period, that indicates the end of a COBOL class definition. The end class header is:

```
END CLASS class-name.
```

***end method header.** A combination of words, followed by a separator period, that indicates the end of a COBOL method definition. The end method header is:

```
END METHOD method-name.
```

* **end of Procedure Division.** The physical position of a COBOL source program after which no further procedures appear.

* **end program header.** A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program header is:

```
END PROGRAM program-name.
```

* **entry.** Any descriptive set of consecutive clauses terminated by a separator period and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, or DATA DIVISION of a COBOL program.

* **environment clause.** A clause that appears as part of an ENVIRONMENT DIVISION entry.

ENVIRONMENT DIVISION. One of the four main component parts of a COBOL program, class definition, or method definition. The ENVIRONMENT DIVISION describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

environment-name. A name, specified by IBM, that identifies system logical units, printer and card punch control characters, report codes, and/or program switches. When an environment-name is associated with a mnemonic-name in the ENVIRONMENT DIVISION, the mnemonic-name may then be substituted in any format in which such substitution is valid.

environment variable. Any of a number of variables that describe the way an operating system is going to run and the devices it is going to recognize.

execution time. 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 COBOL/370 Release 1 bootstrap routine. It must be link-edited with any module that contains a COBOL/370 Release 1 program.

IGZCBSO. The COBOL for MVS and VM Release 2 and COBOL for OS/390 and VM bootstrap routine. It must be link-edited with any module that contains a COBOL for MVS and VM Release 2 or COBOL for OS/390 and VM program.

* **imperative statement.** A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

* **implicit scope terminator.** A separator period which terminates the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

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.

interface. The information that a *client* must know to use a *class*—the names of its *attributes* and the signatures of its *methods*. With direct-to-SOM compilers such as COBOL, the interface to a class may be defined by native language syntax for class definitions. Classes implemented in other languages might have their interfaces defined directly in SOM Interface Definition Language (IDL). The COBOL compiler has a compiler option, IDLGEN, to automatically generate IDL for a COBOL class.

Interface Definition Language (IDL). The formal language (independent of any programming language) by which the *interface* for a class of *objects* is defined in a IDL file, which the SOM compiler then interprets to create an implementation template file and binding files. SOM's Interface Definition Language is fully compliant with standards established by the Object Management Group's Common Object Request Broker Architecture (*CORBA*).

interlanguage communication (ILC). The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages.

intermediate result. An intermediate field containing the results of a succession of arithmetic operations.

*** internal data.** The data described in a program excluding all external data items and external file connectors. Items described in the LINKAGE SECTION of a program are treated as internal data.

*** internal data item.** A data item which is described in one program in a run unit. An internal data item may have a global name.

internal decimal item. A format in which each byte in a field except the rightmost byte represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. (Also known as packed decimal.)

*** internal file connector.** A file connector which is accessible to only one object program in the run unit.

*** intra-record data structure.** The entire collection of groups and elementary data items from a logical record which is defined by a contiguous subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

intrinsic function. A predefined function, such as a commonly used arithmetic function, called by a built-in function reference.

* **invalid key condition.** A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

* **I-O-CONTROL.** The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

* **I-O-CONTROL entry.** An entry in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION which contains clauses that provide information required for the transmission and handling of data on named files during the execution of a program.

* **I-O-Mode.** The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phase for that file.

* **I-O status.** A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

iteration structure. A program processing logic in which a series of statements is repeated while a condition is true or until a condition is true.

K

K. When referring to storage capacity, two to the tenth power; 1024 in decimal notation.

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, PL/I, and Java applications compiled by Language Environment-conforming compilers.

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.

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.

metaclass. A SOM class whose instances are SOM class-objects. The methods defined in metaclasses are executed without requiring any object instances of the class to exist, and are frequently used to create instances of the class.

method. Procedural code that defines one of the operations supported by an object, and that is executed by an INVOKE statement on that object.

* **Method Definition.** The COBOL source unit that defines a method.

* **method identification entry.** An entry in the METHOD-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the method-name and assign selected attributes to the method definition.

* **method-name.** A user-defined word that identifies a method.

* **mnemonic-name.** A user-defined word that is associated in the ENVIRONMENT DIVISION with a specified implementor-name.

multitasking. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks. When running under the Language Environment product, multitasking is synonymous with *multithreading*.

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 file.** A file that is opened in either the OUTPUT mode or EXTEND mode.

* **output mode.** The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

* **output procedure.** A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

overflow condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

P

packed decimal item. See "internal decimal item".

* **padding character.** An alphanumeric character used to fill the unused character positions in a physical record.

page. A vertical division of output data representing a physical separation of such data, the separation being based on internal logical requirements and/or external characteristics of the output medium.

* **page body.** That part of the logical page in which lines can be written and/or spaced.

* **paragraph.** In the Procedure Division, a paragraph-name followed by a separator period and by zero, one, or more sentences. In the IDENTIFICATION and ENVIRONMENT DIVISIONs, a paragraph header followed by zero, one, or more entries.

* **paragraph header.** A reserved word, followed by the separator period, that indicates the beginning of a paragraph in the IDENTIFICATION and ENVIRONMENT DIVISIONs. The permissible paragraph headers in the IDENTIFICATION DIVISION are:

```
PROGRAM-ID. (Program IDENTIFICATION DIVISION)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

The permissible paragraph headers in the ENVIRONMENT DIVISION are:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class
   CONFIGURATION SECTION)
FILE-CONTROL.
I-O-CONTROL.
```

* **paragraph-name.** A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter. 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 header, a user-defined word that identifies a COBOL source program.

* **pseudo-text.** A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

* **pseudo-text delimiter.** Two contiguous equal sign characters (==) used to delimit pseudo-text.

* **punctuation character.** A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
:	colon
.	period (full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
b	space
=	equal sign

Q

QSAM (Queued Sequential Access Method). An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

* **qualified data-name.** An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

* **qualifier.**

1. A data-name or a name associated with a level indicator which is used in a reference either together with another data-name which is the name of an item that is subordinate to the qualifier or together with a condition-name.
2. A section-name that is used in a reference together with a paragraph-name specified in that section.
3. A library-name that is used in a reference together with a text-name associated with that library.

R

* **random access.** An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

* **record.** See "logical record".

* **record area.** A storage area allocated for the purpose of processing the record described in a record description entry in the File Section of the DATA

DIVISION. In the File Section, the current number of character positions in the record area is determined by the explicit or implicit RECORD clause.

* **record description.** See "record description entry".

* **record description entry.** The total set of data description entries associated with a particular record. The term is synonymous with record description.

recording mode. The format of the logical records in a file. Recording mode can be F (fixed-length), V (variable-length), S (spanned), or U (undefined).

record key. A key whose contents identify a record within an indexed file.

* **record-name.** A user-defined word that names a record described in a record description entry in the DATA DIVISION of a COBOL program.

* **record number.** The ordinal number of a record in the file whose organization is sequential.

recursion. A program calling itself or being directly or indirectly called by a one of its called programs.

recursively capable. A program is recursively capable (can be called recursively) if the RECURSIVE attribute is on the PROGRAM-ID statement.

reel. A discrete portion of a storage medium, the dimensions of which are determined by each implementor that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

reentrant. The attribute of a program or routine that allows more than one user to share a single copy of a load module.

* **reference format.** A format that provides a standard method for describing COBOL source programs.

reference modification. A method of defining a new alphanumeric data item by specifying the leftmost character and length relative to the leftmost character of another alphanumeric data item.

* **reference-modifier.** A syntactically correct combination of character-strings and separators that defines a unique data item. It includes a delimiting left parenthesis separator, the leftmost character position, a colon separator, optionally a length, and a delimiting right parenthesis separator.

* **relation.** See "relational operator" or "relation condition".

* **relational operator.** A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

Operator	**Meaning**
IS GREATER THAN	Greater than
IS >	Greater than
IS NOT GREATER THAN	Not greater than
IS NOT >	Not greater than
IS LESS THAN	Less than
IS <	Less than
IS NOT LESS THAN	Not less than
IS NOT <	Not less than
IS EQUAL TO	Equal to
IS =	Equal to
IS NOT EQUAL TO	Not equal to
IS NOT =	Not equal to
IS GREATER THAN OR EQUAL TO	Greater than or equal to
IS >=	Greater than or equal to
IS LESS THAN OR EQUAL TO	Less than or equal to
IS <=	Less than or equal to

* **relation character.** A character that belongs to the following set:

Character	**Meaning**
>	greater than
<	less than
=	equal to

* **relation condition.** The proposition, for which a truth value can be determined, that the value of an arithmetic expression, data item, nonnumeric literal, or index-name has a specific relationship to the value of another arithmetic expression, data item, nonnumeric literal, or index name. (See also "relational operator".)

* **relative file.** A file with relative organization.

* **relative key.** A key whose contents identify a logical record in a relative file.

* **relative organization.** The permanent logical file structure in which each record is uniquely identified by

an integer value greater than zero, which specifies the record's logical ordinal position in the file.

* **relative record number.** The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

* **reserved word.** A COBOL word specified in the list of words that may be used in a COBOL source program, but that must not appear in the program as user-defined words or system-names.

* **resource.** A facility or service, controlled by the operating system, that can be used by an executing program.

* **resultant identifier.** A user-defined data item that is to contain the result of an arithmetic operation.

reusable environment. A reusable environment is when you establish an assembler program as the main program by using either ILBOSTP0 programs, IGZERRE programs, or the RTEREUS 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 Divisions. In the ENVIRONMENT and DATA DIVISIONs, a section header is composed of reserved words followed by a separator period. The permissible section headers in the ENVIRONMENT DIVISION are:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

The permissible section headers in the DATA DIVISION are:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a separator period.

* **section-name.** A user-defined word that names a section in the Procedure Division.

selection structure. A program processing logic in which one or another series of statements is executed, depending on whether a condition is true or false.

* **sentence.** A sequence of one or more statements, the last of which is terminated by a separator period.

* **separately compiled program.** A program which, together with its contained programs, is compiled separately from all other programs.

* **separator.** A character or two contiguous characters used to delimit character-strings.

* **separator comma.** A comma (,) followed by a space used to delimit character-strings.

* **separator period.** A period (.) followed by a space used to delimit character-strings.

* **separator semicolon.** A semicolon (;) followed by a space used to delimit character-strings.

sequence structure. A program processing logic in which a series of statements is executed in sequential order.

* **sequential access.** An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

* **sequential file.** A file with sequential organization.

* **sequential organization.** The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

serial search. A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last.

* **77-level-description-entry.** A data description entry that describes a noncontiguous data item with the level-number 77.

* **sign condition.** The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

* **simple condition.** Any single condition chosen from the set:
  Relation condition
  Class condition
  Condition-name condition
  Switch-status condition
  Sign condition

Single Byte Character Set (SBCS). A set of characters in which each character is represented by a single byte. See also "EBCDIC (Extended Binary-Coded Decimal Interchange Code)."

slack bytes. Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

SOM. See “System Object Model”

* **sort file.** A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

* **sort-merge file description entry.** An entry in the File Section of the DATA DIVISION that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

* **SOURCE-COMPUTER.** The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the source program is compiled, is described.

* **source computer entry.** An entry in the SOURCE-COMPUTER paragraph of the ENVIRONMENT DIVISION which contains clauses that describe the computer environment in which the source program is to be compiled.

* **source item.** An identifier designated by a SOURCE clause that provides the value of a printable item.

source program. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences with the IDENTIFICATION DIVISION or a COPY statement. A COBOL source program is terminated by the end program header, if specified, or by the absence of additional source program lines. 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 (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon

* **special-character word.** A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an ENVIRONMENT DIVISION paragraph in which environment-names are related to user-specified mnemonic-names.

* **special names entry.** An entry in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating implementor-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

* **special registers.** Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of a specific COBOL feature.

* **standard data format.** The concept used in describing the characteristics of data in a COBOL DATA DIVISION under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

* **statement.** A syntactically valid combination of words, literals, and separators, beginning with a verb, written in a COBOL source program.

STL. STL File System: native workstation and PC file system for COBOL and PL/I. Supports sequential, relative, and indexed files, including the full ANSI 85 COBOL standard I/O language and all of the extensions described in *COBOL Language Reference*, unless exceptions are explicitly noted.

structured programming. A technique for organizing and coding a computer program in which the program comprises a hierarchy of segments, each segment having a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the hierarchy.

* **subclass.** A class that inherits from another class. When two classes in an inheritance relationship are considered together, the subclass is the inheritor or inheriting class; the *superclass* is the inheritee or inherited class.

* **subject of entry.** An operand or reserved word that appears immediately following the level indicator or the level-number in a DATA DIVISION entry.

* **subprogram.** See "called program".

* **subscript.** An occurrence number represented by either an integer, a data-name optionally followed by an integer with the operator + or -, or an index-name optionally followed by an integer with the operator + or -, that identifies a particular element in a table. A subscript may be the word ALL when the subscripted identifier is used as a function argument for a function allowing a variable number of arguments.

* **subscripted data-name.** An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

* **superclass.** A class that is inherited by another class. See also *subclass*.

switch-status condition. The proposition, for which a truth value can be determined, that an UPSI switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

* **symbolic-character.** A user-defined word that specifies a user-defined figurative constant.

syntax. The rules governing the structure of a programming language and the construction of a statement in a programming language.

* **system-name.** A COBOL word that is used to communicate with the operating environment.

System Object Model (SOM). IBM's object-oriented programming technology for building, packaging, and manipulating class libraries. SOM conforms to the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standards.

T

* **table.** A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

* **table element.** A data item that belongs to the set of repeated items comprising a table.

text deck. Synonym for *object deck* or *object module*.

* **text-name.** A user-defined word that identifies library text.

* **text word.** A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

- A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for nonnumeric literals. The right parenthesis and left parenthesis characters, regardless of context within the library, source program, or pseudo-text, are always considered text words.
- A literal including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark that bound the literal.
- Any other sequence of contiguous COBOL characters except comment lines and the word 'COPY' bounded by separators that are neither a separator nor a literal.

top-down design. The design of a computer program using a hierarchic structure in which related functions are performed at each level of the structure.

top-down development. See "structured programming".

trailer-label. (1) A file or data set label that follows the data records on a unit of recording medium. (2) Synonym for end-of-file label.

* **truth value.** The representation of the result of the evaluation of a condition in terms of one of two values: true or false.

U

* **unary operator.** A plus (+) or a minus (-) sign, that precedes a variable or a left parenthesis in an arithmetic expression and that has the effect of multiplying the expression by +1 or -1, respectively.

unit. A module of direct access, the dimensions of which are determined by IBM.

universal object reference. A data-name that can refer to an object of any class.

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.

VM/SP (Virtual Machine/System Product). An IBM-licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a "real" machine.

volume. A 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.

Y

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".

Index

Special characters

/ (slash) in CURRENCY-SIGN clause
  changed 140
* (asterisk) 128

Numerics

16-MB line, storage requirements 23
31-bit addressing range 53
64-bit addressing 243

A

A in PICTURE clause 190
abbreviated combined relation conditions
    parenthesis evaluation changed 128
abend codes 86
abends
    caused by unclosed files (C03) 66, 81
    compatible behavior 60
    forcing using ILBOABN0 69, 88
    obtaining after severe errors 51
    OCx, caused by unsupported
      calls 273
    U3504, caused by unsupported
      calls 274
    when using STEPLIB for IMS
      programs 32
abnormal termination exit
    specifying 58
above-the-line storage 23
ABTERMENC runtime option 51
ABTERMENC(ABEND) runtime option
    runtime detected errors, effects of 85
ACCEPT statement
    keyword FROM requirements 128
    system input devices for
      mnemonic-name suboption 160
ACCEPT SYSIN data set behavior 95
accessibility
    of Enterprise COBOL xi
    of this document xii
    using z/OS xi
ACTUAL KEY clause 120
ADDRESS OF special register 214, 216
addressing range, allocating EXTERNAL
  data 53
addressing, based 212
advantages of new compiler and run
  time 6
AFTER phrase of PERFORM 146
AIXBLD runtime option 66, 80
ALL31 runtime option
    use on CICS 54
    use on non-CICS 53
ALPHABET clause 136, 177
ALPHABETIC class 136, 177
AMODE considerations 224
    when assembler invokes COBOL 281
AMODE considerations (continued)
    when upgrading Language
      Environment 106, 108
AMODE, overriding default setting 309
AMODE(24) programs, runtime options
  for 52, 53, 67
ANALYZE compiler option
    not available in Enterprise
      COBOL 171
    not available with Enterprise
      COBOL 12
ANYHEAP runtime option 52, 96
APAR PN32747 66, 81
APAR PN46223 93
APAR PN55178 66, 81
APAR PN63666 94
APAR PN63674 94
APAR PN65736 98
APAR PQ38838 101
APARs
    for initializing runtime
      environment 94
    for link-editing PL/I programs 93
    for unclosed files 66, 81
applications
    comprised of RES programs 104
    enabled through ILC 91
    taking an inventory of (runtime) 26
    taking an inventory of (source) 37
APPLY CORE-INDEX clause 119
APPLY RECORD-OVERFLOW
  clause 120
APPLY REORG-CRITERIA clause 119
Area A, periods in 132, 161, 168
ARITH compiler option
    for converted IBM COBOL
      programs 169
arithmetic accuracy 136
ASCII data set 306
ASMTDLI 322
ASRA abend failure symptom 274
assembler driver 278
assembler programs
    AMODE requirements when
      invoking 281
    call considerations
      requirements 271
      restrictions for SORT or
        MERGE 69
      supported calls on CICS 274
      supported calls on non-CICS 273
    calling ILBOSTP0 67
    causing C03 abends with unclosed
      files 66, 81
    changing program mask 276
    closing files 81
    DL/I CALL interface routine 213
    dynamic calls to 109
    effects of calling ILBOSTP0 62, 74
    effects of user-written error handling
      routines 275
assembler programs (continued)
    effects of using SVC LINK 63, 76, 83
    enclave termination restrictions 66,
      81
    finding COBOL TGT 226
    link-edit requirements 62, 74
    loading and BALRing COBOL 280
    loading and calling COBOL 280
    loading and deleting COBOL 280
    paragraph name restrictions 137
    save area requirements 271
    subpools, storage in 281
assembler user exit and return codes 60
ASSIGN ... FOR MULTIPLE REEL/UNIT
  phrase 121
ASSIGN ... OR clause 121
ASSIGN clause 137
ASSIGN TO integer system-name
  clause 121
assistive technologies xi
asterisk (*) 128
ATTACH SVC, effect on parameter list
  processing 279
attributes, complexity ratings 28

B

B in PICTURE clause 137, 190
BALR instruction, for assembler
  programs 271
base addressability
    CICS program changes 212
    examples
      CICS chained storage area 216
      CICS communications area 214
      processing storage areas exceeding
        4K 215
      when using OCCURS
        DEPENDING ON 216
basic mapping support, CICS 213
batch applications
    differences with debugging 284
    recommended runtime options
      for 52
BATCH compiler option 154
batch debugging 284
BDAM files 120
BELOW storage
    considerations when upgrading
      Language Environment 105
below-the-line storage 23
BELOWHEAP runtime option 52, 96
benefits of new compiler and run time 6
binary zeros, initializing
  WORKING-STORAGE to 53
BLANK WHEN ZERO clause 128
BLDL user exit 319
BLL cells
    automated conversion of 268
    base addressability 212
    CICS chained storage areas 216

© Copyright IBM Corp. 1991, 2006

BLL cells *(continued)*
removed 213
when not explicitly defined 214
bootstrap routine
for Enterprise COBOL 103
for IBM COBOL 103
for NORES behavior 103
for VS COBOL II 220
BUF compiler option 153
buffer size specification 153
BUFSIZE compiler option
for converted OS/VS COBOL programs 153

C

C
ILC with OS/VS COBOL 71
C/370
ILC with VS COBOL II programs 93
CALL statement
changes for USING phrase 137
ON OVERFLOW, CMPR2/NOCMPR2 178
programs processed by CICS translator 221
callable services
CEE3ABD 69, 88
CEEMRCR 276
CEETEST 284
calls
between OS/VS COBOL and VS COBOL II 75
DL/I interface 213
dynamic to alternate entry points 140
improving CICS performance 55
requirements
assembler programs 271
OS/VS COBOL and Enterprise COBOL 220, 222
OS/VS COBOL and FORTRAN 63
OS/VS COBOL and PL/I 63, 76
static calls from Enterprise COBOL programs 219
restrictions
assembler programs 69
dynamic on OS/VS COBOL 63
recursive 276
when using Language Environment preinitialization 95
SOM services, to 166
static, between VS COBOL II and Enterprise COBOL 226
supported
on CICS 221, 274
on non-CICS 273
CBLOPTS runtime option 51
CBLPSHPOP runtime option 55, 98
CBLQDA runtime option 52
CBLTDLI 210, 322
CCCA conversion tool
BDAM file conversion 120
brief description 117
detailed description 267
ISAM file conversion 119
CD FOR INITIAL INPUT 121
CEE3ABD callable service 69, 88
CEEBX05A 86
CEEBX05A sample COBOL user exit 60
CEECOPT default runtime options CSECT 77
CEEDOPT default runtime options CSECT 64, 77
CEEDUMP 68, 87
considerations when upgrading Language Environment 106
dump ouput destination 69
CEEMRCR callable service 276
CEEPIPI 95
CEEROPT runtime options CSECT
interaction with CEEUOPT, CEEDOPT 65, 78
CEETDLI 322
CEETEST callable service 284
CEEUOPT default runtime options CSECT
applications that can use 78
coexisting with IGZEOPT 79
CEEWUCHA
effects when using 86
runtime detected errors, effects of 86
chained storage area, CICS example 216
changes to compiler, summary xiv
CICS
abnormal termination exit 58
basic mapping support (BMS) 213
call considerations
DL/I CALL interface 213
dynamic calls by OS/VS COBOL 63
HANDLE commands 98
processed by CICS translator 221
static CALL statement 221
supported under Language Environment 274
converting source programs
automatically (CCCA) 268
containing BLL cells 212
DATE special register 122
handling commands, compatibility 98
LENGTH OF special register 213
macro-level to command-level 269
SERVICE RELOAD statement 212
DISPLAY statement 100
effect of TRUNC compiler option 210
integrated translator 211
invocation changes 57
Language Environment output considerations
default destination 57
message handling 84
obtaining a system dump 58
migrating separate translator to integrated translator 210
OS/VS COBOL programs, support for 71, 113, 209
performance considerations 97
programs requiring upgrade 63, 75
releases supported 209
CICS *(continued)*
required compiler options
CICS 210
DATA(24) 210
LIB 210
NODYNAM 210
RENT 210
running NORENT programs 98
runtime options considerations
effects of using IGZEOPT 79
fixing runtime options 77
other 55
recommended 54
transaction dump
obtaining 58
output 68
virtual storage usage 24
WORKING-STORAGE limits on 97
CICS Application Migration Aid 269
CICS compiler option 10, 210, 212
CICS integrated translator 211
benefits of 211
CBL/PROCESS statements, considerations for 211
comment lines, considerations for 211
DFHCOMMAREA considerations 211
migrating from separate translator 210
TRUNC compiler option considerations 211
using SIZE(MAX) 211
CICS TS (Transaction Server) 71
CLISTs changes required 56
CLOSE statement
DISP phrase unsupported 121
FOR REMOVAL phrase 129
POSITIONING phrase 121
CMPR2 compiler option
ALPHABET clause 177
ALPHABETIC class 177
CALL...ON OVERFLOW class 178
COPY statement 182
COPY...REPLACING statement 180
definition for 175
EXIT PROGRAM 184
file status codes 183
for converted VS COBOL II programs 164
language differences from NOCMPR2 176
not available with Enterprise COBOL 12
PERFORM statement 186
PERFORM...VARYING...AFTER 188
PICTURE clause 190
PROGRAM COLLATING SEQUENCE 192
READ INTO and RETURN INTO 193
RECORD CONTAINS n CHARACTERS 194
reserved words 195
scaled integers and nonnumerics 179
SET...TO TRUE 196

CMPR2 compiler option *(continued)*
SIZE ERROR on MULTIPLY and DIVIDE 198
UNSTRING statement 199
upgrading programs compiled with 175
upgrading VS COBOL II programs compiled with 157
UPSI switches 205
variable-length group moves 206
variable-length records 195
COBOL 68 Standard 113
COBOL 85 Standard
interpretation changes 158
tools for converting source programs to 263
COBOL and CICS/VS Command Level Conversion Aid
detailed description 267
ISAM file conversion 119
COBOL applications
taking an inventory of (run time) 27
taking an inventory of (source) 37
COBOL for MVS & VM
bootstrap routine 103
upgrading to Enterprise COBOL 165
COBOL for OS/390 & VM
bootstrap routine 103
upgrading to Enterprise COBOL 165
COBOL/370
bootstrap routine 103
upgrading to Enterprise COBOL 165
COBTEST 283, 285
CODE-SET clause, FS 39 305
codes, abend 60
command-level CICS programs, converting to 269
COMMAREA considerations 213
comment lines
in VS COBOL II programs 158
communication feature 121
comparing group to numeric packed-decimal item 129
compatibility
abend codes 60
alternatives to LIBKEEP 94
recommended runtime options for non-CICS 51
recommended runtime options for on CICS 54
reusable runtime environment 67, 81
SORT or MERGE considerations 69, 89
compilation
Report Writer programs 118
compile
OS/VS COBOL programs, requiring upgrade to 63
VS COBOL II programs, benefits of 75
compiler limits 301
compiler options
complete list 289
for compiling VS COBOL II programs 163
for converted OS/VS COBOL programs 153
compiler options *(continued)*
for OS/VS COBOL, not supported 154
for SOM-based object-oriented COBOL, not supported 166
required for CICS integrated translator 212
upgrading from IBM COBOL 169
using TEST(SYM) 87
complexity ratings
conversion priorities relating to 40
conversion priority 38
program attributes 28
condition handling 53
condition handling on IMS 322
conditions, obtaining abends after 51
conversion priority
complexity ratings relating to 40
conversion strategies
incremental conversion 11
moving to Language Environment 23
upgrading source 35
conversion tools
CICS Application Migration Aid 36, 269
CMPR2 compiler option 36, 118
COBOL Conversion Tool (CCCA) 36, 117, 267
Debug Tool Load Module Analyzer 270
Edge Portfolio Analyzer 270
FLAGMIG compiler option 36, 118
MIGR compiler option 36, 117, 263
NOCOMPILE compiler option 36
Report Writer Precompiler 36, 269
vendor products 270
converting source
IBM COBOL programs, requiring 165
OS/VS COBOL CICS programs 213
OS/VS COBOL programs, requiring 63
recommended callable service 88
scenarios
Report Writer discarded 44
Report Writer retained 45
with CICS 42
without CICS or report writer 41
tasks when updating 45
VS COBOL II programs, benefits of 75
VS COBOL II programs, requiring 157
COPY statement 139
COPY statement, using @, #, $ 182
COPY...REPLACING statement 180
COUNT compiler option 154, 283
CURRENCY-SIGN clause 140
CURRENT-DATE special register 121

D

DASD storage 23
data base integrity, ensuring 322
DATA DIVISION, two periods in a row 132
data sets, differences with Language Environment preinit 95
data-name, unique compared to program-id 133
DATA(24) compiler option
CICS program requirements 210
or converted OS/VS COBOL programs 153
OS/VS COBOL program requirements 220
DATA(31) compiler option
change in behavior for 105
DATE special register 122
DB2
coprocessor considerations 316
coprocessor integration 315
coprocessor, benefits of 315
separate precompiler 315
DD definitions 90
ddnames
MSGFILE restrictions 53
required for output 56
SYSPRINT 80
DEBUG runtime option 66, 80
Debug Tool 4, 283
Debug Tool Load Module Analyzer 270
debugging
existing applications 283
initiating the Debug Tool 284
language comparison 285
OS/VS COBOL output 70, 90
upgraded applications 283
when debug data not produced 69
DEBUGGING declarative 148
decimal overflow, program mask and 276
declaratives
debugging changes 148
GIVING phrase of ERROR 124
default destination, Language Environment output 56
default runtime options
preventing programmers from changing 64, 77
recommended for non-CICS 51
recommended for on CICS 54
defaults
application-specific
CEEUOPT 65, 78
installation-wide
CEEDOPT 64, 77
on CICS, CEECOPT 64
under CICS, CEECOPT 77
region-wide
CEEROPT 65, 78
DFHCOMMAREA
CALL statement considerations 221
integrated CICS translator, considerations for 211
DFHEIBLK 221
DIAGTRUNC compiler option
for converted OS/VS COBOL programs 153
DISK file output 70, 90
DISP phrase of CLOSE 121
DISPLAY output
impacts to ILC applications 92

DISPLAY statement 123
CICS considerations 100
DISPLAY SYSOUT output
data set behavior with Language Environment preinitialization 95
sent to file with RECFM=FB 70
DISPLAY SYSPUNCH data set behavior 95
DISPLAY UPON SYSOUT
differences between Language Environment and VS COBOL II 90
output with RECFM=FB 90
DIVIDE statement 145, 198
DL/I CALL interface 213
DSA, using to find TGT 226
dump format changes
problem determination 68
DUMP macro 106
dumps
compiled with the FDUMP option 87
destination
indicating by changing ddname 56
on CICS 57
differences with Language Environment 87
formatted dumps, Language Environment 87
obtaining system or transaction 58
when not produced for SORT or MERGE 69
dynamic allocation, QSAM (CBLQDA) 52
dynamic calls
CICS considerations
restrictions for OS/VS COBOL programs 63
supported under Language Environment 274
when allowed under Enterprise COBOL 221
compiler option requirement 220
difference in behavior for RENT programs 93, 94, 280
placed to alternate entry points 140
supported on non-CICS under Language Environment 273

E

Edge Portfolio Analyzer 270
education
available for Enterprise COBOL 36
available for Language Environment 26
enclave
multiple enclaves,restrictions in OS/VS COBOL 63
enclave boundary with assembler programs 272
enclave termination
closing OS/VS COBOL programs prior to 66
closing VS COBOL II programs prior to 81
ENDJOB compiler option 154, 320
Enterprise COBOL
advantages of 6
bootstrap routine 103
changes with 12
CICS considerations
upgrading OS/VS COBOL CICS programs 213
compiler limits 301
compiler options, complete list 289
compiler options, unsupported 163
high level overview 3
installing, documentation needed 35
logical record length 160
new reserved words 167
prolog format changes 155
recommended callable service 88
reserved words, complete list 245
runtime considerations 226
upgrading IBM COBOL programs to 14
upgrading OS/VS COBOL programs to 14
upgrading VS COBOL II programs to 14
Enterprise COBOL programs
existing applications, adding to 219
link-edit considerations, when adding 219, 221
link-editing considerations 219
NORES programs, considerations for 221
RES programs, considerations for 219
restrictions for CICS 219
ENTRY points 140
ENVIRONMENT DIVISION, two periods in a row 132
error handling routines, user-written 275
error messages, trace entry changes 70
error, insufficient storage 168
errors
finding information on using system dump 68, 88
obtaining abends after 51
subscripts out of range message 149
ESPIE exit, conversion requirements 275
ESTAE exit, conversion requirements 275
evaluation changes in relation conditions 138
EVENTS compiler option
not available in Enterprise COBOL 171
EXAMINE statement 122
EXEC CICS LINK
communicating with other languages 71
invoking SORT statement 97
support under Language Environment 274
EXEC CICS statement 211
EXEC CICS XCTL
communicating with other languages 71
EXEC DLI statement 211
EXHIBIT output 70, 90
EXHIBIT statement 123
existing applications
adding Enterprise COBOL programs to 219
ensuring compatibility for 51
link-editing 57
preventing file status 39 305
specifying correct runtime library 56
specifying runtime options for (OS/VS COBOL) 64
specifying runtime options for (VS COBOL II) 76
using ILC 71
EXIT PROGRAM statement 140
differences between CMPR2 and NOCMPR2 184
exponent underflow, program mask and 276
exponentiation changes 136
Extended Link Pack Area (ELPA) 212
extensions, undocumented 128, 161, 168
EXTERNAL data allocation 53
External names, changed in Enterprise COBOL 167

F

FD support in REDEFINES clause 134
FDUMP compiler option
mapped to TEST 164
receiving similar output 283
VS COBOL II programs compiled with 87
feedback code 60
file handling
changes when upgrading Language Environment 106
file status 39
avoiding when processing new files 306
preventing for QSAM files 305
preventing for VSAM files 125
FILE STATUS clause 140
file status codes, CMPR2/NOCMPR2 183
FILE-CONTROL paragraph
FILE STATUS clause changed 140
FILE-LIMIT clause unsupported 124
files
output file space 108
preventing file status 39 305
QSAM, dynamic allocation (CBLQDA) 52
unclosed causing abends 66, 81
fixed runtime options under Language Environment 64, 77
fixed-length records, defining 306
fixed-point overflow, program mask and 276
FLAGMIG compiler option
definition for 176
not available with Enterprise COBOL 12, 164
FLAGSAA compiler option 164
floating-point changes 136
FLOW compiler option 283
flow of control, ended 129, 184

FLOW runtime option 66
FOR REMOVAL phrase of CLOSE statement 129
Format-x (F,S,U,V) files 305
formatted dumps
  generated by Language Environment 68, 87
    dump destination (CEEDUMP) 69, 87
    dump destination under CICS 69, 88
FORTRAN
  ILC with OS/VS COBOL programs 63, 71
  ILC with VS COBOL II programs 92
FROM, requirements with ACCEPT statement 128

G

GENERATE statement 118
GOBACK statement 129, 140
  combined with SORT or MERGE, effects of 89
  differences between CMPR2 and NOCMPR2 184

H

hardware detected errors, intercepting 53
HEAP runtime option 52, 96, 105

I

IBM COBOL
  undocumented extensions for 168
  upgrading source, requiring 14, 165
  upgrading to Enterprise COBOL 165
IDCAMS REPRO facility 120
IDLGEN compiler option
  not supported in Enterprise COBOL 166
IF statement 142
IGYWAPSX, for parameter list processing 279
IGZ prefixed messages, how managed 84
IGZ0005S 273
IGZ0014W, suppressing 98
IGZ0079S 274
IGZBRDGE object module 74, 103
IGZCBSN bootstrap routine 75, 103, 220
IGZCBSO bootstrap routine 75, 103
IGZCFCC, change to RMODE 109
IGZEBST bootstrap routine 220
IGZEOPD runtime options module 79
IGZEOPT and MSGFILE 98
IGZEOPT runtime options module 79
IGZEPSX, for parameter list processing 279
IGZERRE routine
  changes in return codes, after using 83
  for upgrading assembler driver 278
IGZERREO CSECT routine 67, 82
IGZETUN and MSGFILE 98
IGZTUNE 96
IGZxxxx routines, replacing 311
IKF prefixed messages 84
ILBOABN0
  forcing an abend 69, 88
  obtaining a system dump 88
ILBOSRV
  when adding Enterprise COBOL, considerations for 226
ILBOSTP0
  assembler driver, alternatives for 278
  link-edit requirements for OS/VS COBOL 62
  link-edit requirements for VS COBOL II 74
  required runtime options for 67
ILBOSTP0 routine
  using to initialize reusable environment, effects of 83
ILBOxxxx routines, replacing 311
ILC
  applications enabled 91
  C/370 and VS COBOL II programs 93
  conversion requirements 63, 76
  FORTRAN and OS/VS COBOL programs 63
  FORTRAN and VS COBOL II programs 92
  general considerations 92
  link-edit requirements for 75
  PL/I and OS/VS COBOL programs 63
  PL/I and VS COBOL II programs 93
  support for existing applications 71
IMS
  BLDL user exit not available 319
  cautions when installing Language Environment 32
  cautions when using reusable environment 82
  ENDJOB compiler option requirements 320
  preloading recommendations 321
  release enabled for CEETDLI 322
  relevant compiler options for 319
index names
  qualified 130
INHERITS clause 166
initial values in WORKING-STORAGE 53
INITIATE statement 118
INSPECT statement
  EXAMINE statement 122
  TRANSFORM statement 127
installation
  cautions for using STEPLIB for IMS programs 32
  compiler, documentation needed 35
  fixing runtime options during 64, 77
  Language Environment, documentation needed 23
  LNKLST/LPALST restrictions 30
INTDATE compiler option
  for converted IBM COBOL programs 170
integrated CICS translator 211
  available with Enterprise COBOL 13
  required compiler options 212
integrated DB2 coprocessor 315
integrated SQL coprocessor 315
interlanguage communication (ILC)
  applications enabled 91
  conversion requirements 63, 76
  link-edit requirements for 75
  support for existing applications 71
intermediate results changed 145
inventory of applications
  Debug Tool Load Module Analyzer 270
  Edge's Portfolio Analyzer 270
  for moving run time to Language Environment 26
  for upgrading source to Enterprise COBOL 37
  WebSphere Studio Asset Analyzer 267
invocation
  procedures for non-CICS applications 56
  recommended runtime option for compatibility 51
  specifying runtime options 65, 78
  with MVS ATTACH 279
INVOKE statement 166, 167
IS evaluation in relation conditions changed 139, 145
ISAM files 119

J

job control language (JCL)
  required changes 56
JUSTIFIED clause 143

K

keyboard navigation xi

L

LABEL RECORD clause 130
LABEL RECORDS clause 124
labels, when redundant for CICS 213
LANGLVL compiler option
  unsupported 154
LANGLVL(1) compiler option
  /, =, and L characters 140
  ACCEPT MESSAGE COUNT 121
  combined abbreviated relational conditions 138
  COPY statement with associated names 139
  DELIMITED BY ALL 149
  JUSTIFIED clause 143
  NOT phrase 138
  PERFORM statement 148
  RESERVE clause 147
  scaling change 143
  SELECT OPTIONAL clause 148

language elements
changed
OS/VS COBOL 136
SOM-based object-oriented COBOL 167
not supported
OS/VS COBOL 119, 121
SOM-based object-oriented COBOL 166
Language Environment
advantages of 6
changes with 12
compatibility factors 51
complexity ratings for moving to 28
dynamic calls to assembler programs 109
formatted dumps 68, 87
installation, general information on 23
link-edit library 57
moving to 13
output file space 108
phasing in using LNKLST/LPALST or STEPLIB 31
runtime library 57
runtime options compared to OS/VS COBOL runtime options 66
runtime options, recommended on CICS 54
runtime options, recommended on non-CICS 51
specifying runtime options
order of precedence 65, 78
specifying runtime options for
OS/VS COBOL programs 64
VS COBOL II programs 76
strategy for moving to 23
trace output, compared to OS/VS COBOL 70, 90
upgrading releases of 105
considerations for 105, 106, 108, 110
using SORT or MERGE 89
Language Environment output
default destination under non-CICS 56
destination under CICS 57
when dumps not produced (SORT or MERGE) 69
Language Environment-conforming assembler programs 278
LANGUAGE runtime option 80
last-used state, when entered in 280
LENGTH OF special register 213
LIB compiler option 210, 212
LIBKEEP runtime option 94
alternatives to LIBKEEP 94
not supported 80
library
specifying correct run time 56
library name, Language Environment 57
Library Routine Retention (LRR) facility 95
library routines, preloading 32
LIBSTACK runtime option 52, 96
LINE-COUNTER special register 118
Link Pack Area (LPA) 212
link-edit
behavior 222
effects on NORES programs 103
example 311
library name for Language Environment 57
overriding default AMODE setting 220, 222
programs requiring
adding Enterprise COBOL programs 219
from OS/VS COBOL run time 62
from VS COBOL II run time 74
reusable environment requirements
OS/VS COBOL programs 67
VS COBOL II programs 81
specifying runtime options after
OS/VS COBOL programs 64
VS COBOL II programs 77
support for multiple load module applications 222
link-edit override 220, 222
LINKAGE SECTION
addressability in CICS 215
BLL cells 214
CICS chained storage areas 216
CICS considerations 213
CICS OCCURS DEPENDING ON example 216
DL/I CALL interface 213
LIST compiler option 155, 164
LISTER features, unsupported 155
LNKLST/LPALST 30
load module analysis
Debug Tool Load Module Analyzer 270
Edge Portfolio Analyzer 270
load modules
inventory of, using conversion tool 270
load modules, multiple 222
LOAD/BALR calls supported under Language Environment 273
LRR facility 95

M

macro-level CICS programs, converting 269
main program
initializing the runtime environment for 94
invocation compatibility 51
message IGZ0005S 273
message IGZ0079S 274
messages
default destinations on CICS 57
IGZ prefixed, format changes 84
MIGR, missing for RENAMES 134
MSGFILE ddname restrictions 53
METACLASS clause 166
METHODS, changed in Enterprise COBOL 167
METHODS, not supported in Enterprise COBOL 166
MIGR compiler option
conversion tool 117, 263
MIGR compiler option *(continued)*
message missing for RENAMES 134
migrating CICS translator
from separate to integrated 210
migrating from CMPR2 to NOCMPR2 175
migrating source
OS/VS COBOL CICS programs 213
OS/VS COBOL programs, requiring 63
scenarios
Report Writer discarded 44
Report Writer retained 45
with CICS 42
without CICS or report writer 41
tasks when updating 45
VS COBOL II programs, benefits of 75
migration strategies
incremental conversion 11
moving to Language Environment 23
upgrading source 35
migration tools
CICS Application Migration Aid 269
COBOL and CICS/VS Conversion Aid (CCCA) 267
Debug Tool Load Module Analyzer 270
Edge Portfolio Analyzer 270
Report Writer Precompiler 269
vendor products 270
MIXRES runtime option
link-edit requirements when used 74
not supported 80
mnemonic-name of system input devices in ACCEPT statement 160
modules, preloading under IMS 321
MOVE ALL statement
to PIC 99 131
MOVE statement
CORRESPONDING changes 131
moving fullword binary items 130
multiple TO specification 131
scaling change 143
SET...TO TRUE 196
warning message for numeric truncation 131
MSGFILE runtime option
changing ddname for messages and reports 56
ddname restrictions 53
MSGFILE, suppressing messages 98
multilanguage conversion 30
multiple enclave restrictions (OS/VS COBOL) 63, 76
multiple load module applications
effects of link-edit with Language Environment 103
effects of using MIXRES 74
supported under Language Environment 222
multiple load modules
OS/VS COBOL considerations 222
VS COBOL II considerations 224
MULTIPLY statement 145, 198
MVS ATTACH, invoking COBOL programs 279

N

national extension characters 182
nested enclaves restrictions (OS/VS COBOL) 63, 76
NOCMPR2 compiler option
  definition for 175
  language differences from CMPR2 176
NOCMPR2 programs
  QSAM dynamic allocation (CBLQDA) 52
  tools for converting source to 263
NOCOMPILE compiler option 154
NODYNAM compiler option 210, 212
NOMINAL KEY clause 119
non-COBOL programs
  ILBOSTP0 support for 67
  unclosed files 66
  unclosed files in 81
nonnumerics, CMPR2/NOCMPR2 179
nonoverridable runtime options 64, 77
nonunique program-id names 133
NORENT compiler option
  above the line support 10
  implications of static calls 220, 222
NORENT programs 98
NORES compiler option 154
  unsupported in Enterprise COBOL 164
NORES programs
  acting as RES-like, considerations for 104
  effects of link-edit with Language Environment 62, 74, 103
  ILC with FORTRAN, considerations for 93
  requiring link-edit with Language Environment 62, 74
  runtime option requirements 53
  specifying runtime options
    from OS/VS COBOL run time 64
    from VS COBOL II run time 77
  specifying runtime options for VS COBOL II using IGZEOPD 79
NOT phrase 138
NOTE statement 125
NSYMBOL compiler option
  for converted IBM COBOL programs 170
NUMCLS compiler option
  for converted OS/VS COBOL programs 153
numeric-edited, differences 133
NUMPROC compiler option
  for converted OS/VS COBOL programs 153

O

OBJECT COMPUTER paragraph 192
object module, prolog format 155, 164
object-oriented COBOL, SOM-based
  compiler options not supported 166
  language elements changed 167
  language elements not supported 166
object-oriented COBOL, SOM-based *(continued)*
  not supported in Enterprise COBOL 12, 165
OBJECTS, changed in Enterprise COBOL 167
OCCURS clause 132
OCCURS DEPENDING ON clause
  changes in values for receiving items 144
  CICS example 216
  RECORD CONTAINS n CHARACTERS 133
  variable-length group moves 206
OCx abends 273
ODO objects, changes for variable-length groups 159
ON SIZE ERROR phrase 145
ON statement 125
OPEN statement
  COBOL 68 support dropped 125
  REVERSED phrase changed 132
operating system detected errors, intercepting 53
OPT compiler option
  for converted OS/VS COBOL programs 153
options
  compiler
    complete list 289
    for IBM COBOL programs 169
    for OS/VS COBOL programs 153
    for VS COBOL II programs 163
  runtime
    recommended for non-CICS 51
    recommended for on CICS 54
    specifying for OS/VS COBOL programs 64
    specifying for VS COBOL II programs 76
ORGANIZATION clause 119, 120
OS/390
  dump output destination 88
  invocation changes 56
  Language Environment library for 57
  Language Environment output default destination 56
  obtaining a system dump 58
  problems with unclosed files 66
  specifying ddname SYSPRINT on 80
  specifying runtime options on 65, 78
OS/VS COBOL
  ALPHABET clause changed 136
  arithmetic accuracy 136
  ASSIGN clause changed 137
  ASSIGN TO integer system-name clause 121
  CALL statement changed 137
  compiler limits 301
  compiler options, complete list 289
  considerations when compiling 153
  CURRENCY-SIGN clause changed 140
  IF statement changed 142
  intermediate results changed 145
  JUSTIFIED clause 143
OS/VS COBOL *(continued)*
  OCCURS DEPENDING ON clause 144
  ON SIZE ERROR phrase changed 145
  PERFORM statement changes 146
  PROGRAM COLLATING SEQUENCE clause 146
  READ statement changes 146
  RERUN clause changes 147
  RESERVE clause changes 147
  reserved word list
    complete list 245
  RETURN statement changes 146
  runtime options compared to Language Environment runtime options 66
  scaling changed 143
  SEARCH statement changes 147
  segmentation changes 148
  SELECT OPTIONAL clause 148
  SORT special register differences 148
  source language debugging 148
  subscripts out of range 149
  trace ouput compared to Language Environment 70
  trace output sequence 90
  undocumented extensions for 128
  unsupported compiler options 154
  upgrading source 14
  UPSI switch evaluation changed 150
  VALUE clause 150
  VSAM files 141
  WHEN-COMPILED 151
  WRITE AFTER POSITIONING statement 151
OS/VS COBOL programs
  base addressability considerations 212
  CICS considerations
    DL/I call interface 213
    support for 71, 113, 209
    upgrading to Enterprise COBOL CICS programs 213
  ILC
    with FORTRAN 63
    with PL/I 63
  multiple enclave restrictions 63, 76
  multiple load modules, combinations of 222
  requiring link-edit with Language Environment 62
  RES programs, ILC with FORTRAN 92
  specifying Language Environment runtime options 64
  symbolic dumps for 68
  upgrading source, requiring 63, 75
  VS COBOL II programs, calling 75
  with user error handling routines 275
OSDECK compiler option 155
OUTDD compiler option
  for converted OS/VS COBOL programs 153
output message file 53

P

PAGE-COUNTER special register 118
paragraph names
  CICS, optional coding changes 213
  error for period missing in 133
  requirements for Enterprise COBOL 133, 137
  restrictions for USING phrase 137
parameter list, processing with MVS ATTACH 279
parameters
  passed by assembler programs 271
  restrictions for paragraph names 137
parenthesis evaluation changed 139
PCB, DL/I CALL interface 213
PERFORM statement
  difference between CMPR2 and NOCMPR2 186
  second UNTIL 132
  VARYING/AFTER options 188
  VARYING/AFTER phrases 146
performance
  CALL statement 220
  improving calls on CICS 98
periods
  missing at end of SD, FD, or RD 133
  missing on paragraph names 133
  multiple in any division 132
  requirements for Area A 132, 161, 168
PGMNAME compiler option 163
  for converted IBM COBOL programs 170
  for converted OS/VS COBOL programs 153
PICTURE clause
  B symbol in 137, 190
  numeric-edited differences 133
  use with VALUE clause 136
PL/I
  ILC with OS/VS COBOL programs 63, 71
  ILC with VS COBOL II programs 75, 93
POSITIONING phrase of CLOSE 121
precedence of runtime options 65, 78
precedence of USE procedures 158
preinitialization 95
preloaded library routines
  causing abends 32
  recommended for IMS 321
preloading programs, for reusable environment 82
PROCEDURE DIVISION, two periods in a row 132
production mode, phasing in Language Environment 30
program attributes
  changes in application behavior 103
  complexity ratings 28
program checks causing ASRA abend 274
PROGRAM COLLATING SEQUENCE clause
  alphabet-name, implicit comparisons 146
PROGRAM COLLATING SEQUENCE clause *(continued)*
  difference between CMPR2 and NOCMPR2 192
program mask, programs that change it 276
program names
  compatibility 153, 163
  requirements 133
  restrictions 56
prolog format 155, 164
PSW information 68, 88

Q

QSAM files
  enabling dynamic allocation (CBLQDA) 52
  preventing files status 39 305
  status key values 140
qualification - using the same phrase repeatedly 133
qualified index names 130
QUEUE runtime option 66, 121

R

R1 requirements for assembler programs 271
R13 requirements for assembler programs 271
R14 requirements for assembler programs 271
READ statement
  implicit elementary MOVEs 146
  INTO phrase, CMPR2/NOCMPR2 193
READY TRACE statement, not supported 126
RECEIVE statement 121
receiving fields, ODO objects 206
RECFM as FB 70, 90
RECFM as FBA
  recommendation for 90
RECFM as U, output file space 108
RECFM as VB 106
recommended runtime options 51, 54
recompiling, CICS programs requiring 75
RECORD CONTAINS n CHARACTERS clause
  difference between CMPR2 and NOCMPR2 194
  when overridden 133
RECORD CONTAINS, fixed-length records 306
RECORDING MODE U 106
records, preventing FS 39 when defining 305
recursive CALLs, restrictions 276
REDEFINES clause
  CICS considerations 213
  FD support dropped 134
  SD support dropped 134
reentrant programs 53
reference modification 159
Register 9 or 13 values 226
Register Save Area (RSA) conventions 226
registers
  PSW information 68, 88
  requirement for assembler programs 272
regression testing
  runtime considerations 33
  source considerations 46
relation condition
  coding changes 134
  evaluation changes 138
REMARKS paragraph 127
RENAMES clause 134
RENT compiler option 210, 212, 220
RENT programs
  difference in behavior for 93, 94, 280
RENT runtime routines 32
REPLACE statement
  affecting EXEC CICS 211
  when required 220
REPLACE statement and comment lines 158
REPORT clause 118
report section 118
Report Writer
  conversion scenario discarding 44
  conversion scenario retaining 45
  conversion tool 118, 269
  language affected 118
Report Writer Precompiler 269
RERUN clause 147
RES compiler option 154, 164
RES programs
  applications, comprised of 104
  implications of RES-like 104
  requiring link-edit with Language Environment 62, 74
  specifying runtime options for
    OS/VS COBOL programs 64
    VS COBOL II programs 77
    VS COBOL II programs using IGZEOPT 79
RESERVE clause 147
reserved words
  comparison of 245
  comparison to VS COBOL II 160
  difference between CMPR2 and NOCMPR2 195
  new in Enterprise COBOL 167
RESET TRACE statement, not supported 126
restrictions
  for non-COBOL programs 66, 81
  for program names 56
  for reusable environment 67, 81
  for unclosed files 66, 81
return codes
  changes when using IGZERRE 83
  ensuring compatibility 60
return routine, assembler programs 272
RETURN statement
  implicit elementary MOVEs 146
  INTO phrase, CMPR2/NOCMPR2 193

reusable environment
cautions under IMS 82
improve performance when using 82
Language Environment, restrictions for 67, 81
link-edit requirements for (OS/VS COBOL) 62
STOP RUN, effects of using 67, 82
REVERSED phrase of OPEN statement 132
RMODE
IGZCFCC, change to 109
RMODE compiler option 220
for converted OS/VS COBOL programs 153
for converted VS COBOL II programs 163
RMODE considerations 224
RMODE link-edit option 309
RPTSTG runtime option 96
RTEREUS runtime option
cautions under IMS 82
comparison between VS COBOL II and Language Environment 80
possible side effects of 52
specifying 84
SVC LINK exceptions 83
using with assembler drivers 278
runtime considerations
adding Enterprise COBOL programs 226
effects of using IGZEOPT 79
inventory of applications 27
managing messages 84
runtime detected errors, timing of abends 85
runtime options
comparisons
between OS/VS COBOL and Language Environment 66
between VS COBOL II and Language Environment 79
Language Environment
ABTERMENC 51
ALL31 53
ANYHEAP 52
BELOWHEAP 52
CBLOPTS 51
CBLPSHPOP 55
CBLQDA 52
HEAP 52
LIBSTACK 52
MSGFILE 53
STACK 52
STORAGE 53
TERMTHDACT 52
TRAP 53
managing storage 52, 96
preventing programmers from changing 64, 77
recommended, under CICS 54
recommended, under non-CICS 51
required setting with AMODE(24) programs 67
specifying
for OS/VS COBOL programs 64
for specific applications 78
runtime options *(continued)*
specifying *(continued)*
for VS COBOL II programs 76
Language Environment order of precedence 65, 78

S

SAMPDAT1 non-CICS sample user exit 58
SAMPDAT2 CICS sample user exit 58
sample source, abnormal termination exit 58
sample user condition handler
CEEWUCHA 86
save area
assembler programs, requirements 271
location when using ILBOABN0 under Language Environment 69
save area, using to find TGT 226
scaled integers, CMPR2/NOCMPR2 179
SCEELKED
link-edit library 57
SCEERUN
runtime library 56, 57
SCEESAMP data set 58
SD support in REDEFINES clause 134
SEARCH statement 147
SEEK statement unsupported 120
segmentation 148
SELECT clause 148
sending fields, ODO objects 206
sequential files 140
SERVICE RELOAD statement
automated conversion of 268
treated as comment 212
upgrading OS/VS COBOL programs, considerations 213
SET option
CICS considerations for 213
SET...TO TRUE, CMPR2/NOCMPR2 196
severe errors
obtaining abends after 51
short on storage (SOS) in CICS regions 54
significance exceptions, program mask and 276
SIMVRD runtime option 80
sixteen-megabyte line 23
SIZE ERROR on MULTIPLY and DIVIDE 198
slash (/) in CURRENCY-SIGN clause changed 140
SOM-based object-oriented COBOL
compiler options not available 166
language elements changed 167
language elements not supported 166
not available with Enterprise COBOL 12, 165
SORT or MERGE
combined with GOBACK statement, effects of 89
considerations 89
in OS/VS COBOL programs 69, 89
in VS COBOL II subprograms 89
SORT or MERGE *(continued)*
undocumented VS COBOL II extensions 101
SORT special registers 148
SORT statement 97
source language conversion
IBM tools 263
inventory of applications 37
strategy for 35
tasks when updating 45
vendor tools 270
space tuning under Language Environment 96
special registers
ADDRESS OF 214
CURRENT-DATE 121
DATE 122
LENGTH OF 213
LINE-COUNTER 118
PAGE-COUNTER 118
PRINT-SWITCH 118
SORT differences 148
TALLY 122
TIME 127
TIME-OF-DAY 127
WHEN-COMPILED 151
SPECIAL-NAMES paragraph 140, 177
SPM instructions 276
SPOUT runtime option
Language Environment synonyms 80
output directed to 97
SQL
coprocessor integration 315
SQL statements
DB2 coprocessor, handling 315
SSRANGE compiler option 149
SSRANGE runtime option 80
STACK runtime option
recommended values for CICS 54
recommended values for non-CICS 52
use for space tuning 96
STACK storage for work area 168
STAE runtime option 80
STANDARD LABEL statement 128
START statement
support changed 127
USING KEY clause unsupported 119, 127
STATE compiler option 154
statement connectors, THEN unsupported 127
static CALL statement
programs supported by Enterprise COBOL on CICS 221
required AMODE override 220, 222
requirements if made from Enterprise COBOL programs 219
supported under Language Environment on CICS 274
supported under Language Environment on non-CICS 273
when RMODE(24) required 220
status key
QSAM files 140
VSAM files 141

STEPLIB
cautions for using with IMS programs 32
example 32
use for phasing in Language Environment 31
STOP RUN statement
communicating with other languages, effects of 92
differences between CMPR2 and NOCMPR2 184
reusable environment, effects of 67, 82
storage management
runtime options for 52, 96
storage reports, ddname requirement 56
storage requirements
compiler 35
with Language Environment 23
STORAGE runtime option 53
storage, in subpools 281
storage, insufficient 168
strategies
incremental conversion 11
run time, moving to 23
upgrading source 35
structure addressing operation not used 213
subpools, storage in 281
subprograms
dynamic calls to ENTRY points 140
using SORT or MERGE in VS COBOL II 89
subroutines, called by assembler driver 278
subscripts 149
SUPMAP compiler option 154
SVC LINK
differences when using under Language Environment 83
effect on assembler programs 63, 76
supported under Language Environment on non-CICS 273
targeting assembler programs 272
SVC LOAD/BALR 280
SVC LOAD/DELETE 280
SXREF compiler option 154
symbolic dumps 68, 87
SYMDMP compiler option 154
SYMDUMP 68
SYSDBOUT 68
SYSLIB, file needed in 57
SYSOUT output
data set behavior with Language Environment preinitialization 95
with RECFM=FB 70, 90
SYSPRINT ddname 80
SYSPUNCH data set behavior 95
system dump
comparisons
between OS/VS COBOL and Language Environment 68
between VS COBOL II and Language Environment 88
obtaining 58
output destination
under OS/VS COBOL 68
system dump *(continued)*
output destination *(continued)*
under VS COBOL II 88
system input devices for mnemonic-name suboption in ACCEPT statement 160
system output ddnames 56
SYSxxxx ddname restrictions 53

T

TALLY special register 122
Task Global Table (TGT) conventions 226
TERMINATE statement 118
terminating statements, required 129
TERMTHDACT runtime option 52, 68
performance consideration 97
specifying 58
TEST compiler option 283
for converted VS COBOL II programs 163
TEST(SYM) compiler option 87
TESTCOB language 285
testing
phasing in Language Environment 30
regression, for run time 33
regression, for source 46
THEN statement 127
TIME-OF-DAY special register 127
trace entries 70
TRACE output 70, 90
TRACK-AREA clause 119
TRACK-LIMIT clause 120
transaction dump 68
TRANSFORM statement unsupported 127
translator option
XOPTS 211
translator, integrated CICS 211
TRAP runtime option
description and recommended setting 53
obtaining list of vendor products enabled 270
TRUNC compiler option
description 298
for CICS applications 210, 211
for converted IBM COBOL programs 170
for converted OS/VS COBOL programs 154
possible differences using TRUNC(OPT) 130
TSO, Language Environment output default destination 56
TYPECHK compiler option
not supported in Enterprise COBOL 166

U

U3504 abends 274
unclosed files 66, 81
undocumented extensions
for IBM COBOL 168
undocumented extensions *(continued)*
for OS/VS COBOL 128
for VS COBOL II 101, 161
UNSTRING statement
coding not accepted 135
difference between CMPR2 and NOCMPR2 199
multiple INTO phrases 136
upgrading
IBM COBOL programs 14
Language Environment 105
OS/VS COBOL programs 14
programs under CICS 75
VS COBOL II programs 14
upgrading source
CICS considerations
DL/I call interface 213
IBM COBOL programs, requiring 165
IBM conversion tools 263
OS/VS COBOL programs, requiring 63
scenarios
Report Writer discarded 44
Report Writer retained 45
with CICS 42
without CICS or report writer 41
strategies for 35
tasks when updating 45
vendor conversion tools 270
VS COBOL II programs, benefits of 75
VS COBOL II programs, requiring 157
UPSI runtime option 66, 80
UPSI switches
difference between CMPR2 and NOCMPR2 205
differences with condition-names 150
USE procedure
precedence in VS COBOL II 158
USE statement
BEFORE STANDARD LABEL 128
DEBUGGING declarative 148
GIVING phrase of ERROR declarative 124
reporting declarative 118
user exits, BLDL 319
user signal conditions, intercepting 53
user written error handling routines 275
Using REXX execs
processing parameter list formats 325

V

VALUE clause
condition-name changes 150
STORAGE runtime option and 53
use with PICTURE clause changed 136
variable-length group moves 206
variable-length group, differences 159
variable-length records, defining 305
VARYING phrase of PERFORM changed 146
VBREF compiler option 155

VBSUM compiler option 154
VCON
  supported COBOL/assembler on CICS 274
  supported COBOL/assembler on non-CICS 273
vendor products
  obtaining a list of 270
  prerequisites for running under Language Environment 26
  prerequisites for using with Enterprise COBOL 37
virtual storage
  factors influencing 23
  on CICS 24
  usage example for non-CICS 24
VS COBOL II
  bootstrap routine 220
  compiler limits 301
  compiler options, complete list 289
  DISPLAY UPON SYSOUT
    compared with Language Environment 90
  formatted dumps
    differences with Language Environment 87
  reserved words, complete list 245
  upgrading source 14
  WORKING-STORAGE limits on CICS 97
VS COBOL II programs
  benefits of compiling with Enterprise COBOL 75
  CBLPSHPOP option examples
    effect on compatibility 100
    no effect on compatibility 99
  compiled with the FDUMP option 87
  multiple load modules, combinations of 224
  NORENT programs 98
  requiring link-edit with Language Environment 74
  requiring upgrade 157
  reserved words, comparison 160
  specifying Language Environment runtime options for 76
  subprograms, using SORT or MERGE 89
  upgrading source programs 157
  using IGZEOPT for programs compiled RES 79
  using WORKING-STORAGE
    differences with Language Environment 93, 94, 280
  with user error handling routines 275
VSAM files
  conversions 119
  status key changes 141

W

WHEN-COMPILED special register 151
WORD(NOOO) compiler option
  for converted IBM COBOL programs 171
WORKING-STORAGE 280
WORKING-STORAGE *(continued)*
  considerations when upgrading Language Environment 105
WORKING-STORAGE data items 224
WORKING-STORAGE limits
  on CICS 97
WORKING-STORAGE section, initial values 53
WRITE statement 151
WSCLEAR runtime option 81

X

XOPTS translator option 211

Z

z/OS
  commonly asked questions and answers 243
  dump output destination 88
  installing Language Environment on 23
  invocation changes 56
  Language Environment link-edit library for 57
  Language Environment output default destination 56
  obtaining a system dump 58
  problems with unclosed files 66, 81
  specifying correct runtime library, under 56
  specifying ddname SYSPRINT on 80
  specifying runtime options on 65, 78
Z's in PICTURE string 133

Readers' Comments — We'd Like to Hear from You

Enterprise COBOL for z/OS
Compiler and Runtime Migration Guide
Version 3 Release 4

Publication No. GC27-1409-05

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
GC27-1409-05

IBM

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department HHX/H3
555 Bailey Avenue
San Jose, CA
United States of America 95141-1099

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Printed in USA

GC27-1409-05