Enterprise COBOL for z/OS and OS/390

Customization Guide

Version 3 Release 2

GC27-1410-02

Enterprise COBOL for z/OS and OS/390

Customization Guide

Version 3 Release 2

GC27-1410-02

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 69.

Second Edition (September 2002)

This edition applies to:

Enterprise COBOL for z/OS and OS/390, Version 3 Release 2 (Program Number 5655-G53)

and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality.

Contents

About this document

This book is for systems programmers who are responsible for customizing Enterprise COBOL for z/OS and OS/390 for their location. This book provides information needed to plan for and customize Enterprise COBOL under z/OS or OS/390. In addition, this book can help to assess the value of Enterprise COBOL to your organization.

In this book, the generic term ″operating system″ is used when referring to z/OS or OS/390. Planning and customizing information specific to the z/OS or OS/390 operating system will be indicated where applicable.

You should have a knowledge of Enterprise COBOL and of your system's operating environment to use this book and ensure a successful customization of Enterprise COBOL.

How to read the syntax diagrams

Throughout this book, syntax for the compiler options is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following table shows the meaning of symbols at the beginning and end of syntax diagram lines.

Symbol	Indicates
>>–	The syntax diagram starts here
–>	The syntax diagram is continued on the next line
>–	The syntax diagram is continued from the previous line
–><	The syntax diagram ends here

  Diagrams of syntactical units other than complete statements start with the >– symbol and end with the –> symbol.

- Required items appear on the horizontal line (the main path).

```
►►──STATEMENT──required item──────────────────────────────────►◄
```

- Optional items appear below the main path.

```
►►──STATEMENT──┬───────────────┬──────────────────────────────►◄
               └─optional item─┘
```

- When you can choose from two or more items, they appear vertically in a stack.

  If you *must* choose one of the items, one item of the stack appears on the main path. The default, if any, appears above the main path and is chosen by the compiler if you do not specify another choice. In some cases, the default is affected by the system in which the program is being run or the environmental parameters specified.

```
             ┌─default-item─────┐
►►──STATEMENT──┼─required choice 1─┼──────────────────────────────────►◄
             └─required choice 2─┘
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
►►──STATEMENT──┬───────────────────┬──────────────────────────────────►◄
             ├─optional choice 1─┤
             └─optional choice 2─┘
```

- An arrow returning to the left above the main line indicates an item that can be repeated.

```
             ┌─,─────────────┐
             ▼               │
►►──STATEMENT────repeatable item─┴──────────────────────────────────►◄
```

  A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.
- Keywords appear in uppercase letters (for example, PRINT). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, `item`). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.
- Use at least one blank or comma to separate parameters.

Using the macro planning worksheets

The planning worksheets in this book will help you prepare to customize Enterprise COBOL. By completing them, you will be able to easily identify those options that you want to change from the IBM-supplied default values. You might also want to use the worksheets as a source from which to actually customize the IBM-supplied default values.

The headings in each worksheet might differ slightly from each other. Refer to the following list of definitions for an explanation of each specific column heading. Worksheets are located on pages 3 and 9.

Option
: The OPTION column identifies the options contained within a specific installation macro. This column represents the options exactly as they are in the macro.

Fixed
: The FIXED column is used to identify the options that can not be overridden by an application programmer. Enter an asterisk [*] into the **Enter * for Fixed** only for those options that you want to be fixed.

Selection
: The SELECTION column is for you to identify the value associated with each option. In the space provided, enter the value you want to assign to each option. Use the topic reference column to locate the specific information about the option that will assist you in selecting the appropriate value.

IBM-Supplied Default
: The IBM-SUPPLIED DEFAULT column identifies the value that is supplied for

the specified installation macro if the option is not altered. If the IBM-supplied default is identical to the value that you desire for installation, you need not modify that option within that specific macro.

Topic Reference
The PAGE REFERENCE column identifies the topic where you will find the syntax diagram for and more specific information about the option.

After you have completed the worksheets, identify those options that are different from the IBM-supplied defaults. These are the items that you must code in the installation macros. The worksheet entries have been positioned such that the order of the entries will remain consistent with the actual coding semantics.

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
  USA
- Fax your comments to this U.S. number: (800)426-7773.
- Use the Online Readers' Comment Form at www.ibm.com/software/ad/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.

Summary of changes

This section lists the major changes that have been made to the Enterprise COBOL for z/OS and OS/390 product and this manual since IBM COBOL for OS/390 & VM Version 2 Release 2. Technical changes are marked in the text by a change bar in the left margin.

Major changes to Enterprise COBOL

Version 3 Release 2, September 2002

- The OPTIMIZE compiler option is now fully supported for programs containing object-oriented syntax for Java interoperability.
- How Unicode and EBCDIC code pages are specified when using SQL statements in COBOL programs has been enhanced:
  - When Unicode host variables (declared with USAGE NATIONAL) are used in SQL statements, it is no longer necessary to specify the Unicode CCSID (1200) explicitly for the variables using the SQL DECLARE VARIABLE statements.
  - The CCSID in effect through the CODEPAGE compiler option now applies to single- or double-byte EBCDIC host variables used in SQL statements without explicit SQL DECLARE VARIABLE statements for the host variables.

Version 3 Release 1, November 2001

- Because of support for POSIX threads and asynchronous signal toleration, an application can contain COBOL programs running in multiple threads within a process. The new compiler option in support of threads and asynchronous signal toleration is:
  - THREAD
- A new national data type, national literals, intrinsic functions, and two new compiler options provide basic run-time support for Unicode. The new compiler option in support of the interoperation of symbols used in literals and PICTURE clauses is:
  - NSYMBOL

  The new compiler option in support of national data types and literals is:
  - CODEPAGE
- Object-oriented syntax now facilitates the interoperation of COBOL and Java programs.
- Basic XML capabilities are added to COBOL.
- The CICS translator has been integrated with the compiler. The new compiler option in support of integrated CICS translation is:
  - CICS

For a history of changes to previous COBOL compilers, see *Enterprise COBOL for z/OS and OS/390 Compiler and Run-Time Migration Guide* .

Chapter 1. Planning to customize Enterprise COBOL

This chapter provides the following information for planning the customization of Enterprise COBOL:
- Making changes after installation—why customize?
- Planning to modify compiler default values
- Planning to place Enterprise COBOL in shared storage
- Planning to create an additional reserved word table

If you're installing IBM Debug Tool, you can decide whether to place its modules in the shared storage and whether you want to set up your CICS® environment to work with Debug Tool.

The information in this chapter helps you plan your customization. For the actual customization procedures, see Chapter 3, "Customizing Enterprise COBOL" on page 53.

This chapter also contains worksheets to help you plan modifications to the IBM-supplied default values within macros. See "Using the macro planning worksheets" on page vi for an explanation of the planning worksheets in this manual.

Important

Confer with the application programmers at your site while you plan the customization of Enterprise COBOL. Doing so will ensure that the modifications you make serve their needs and support the applications being developed.

Making changes after installation—why customize?

When you install Enterprise COBOL, you receive IBM-supplied defaults for compiler options and phases and the reserved word table. You might want to customize Enterprise COBOL to better suit the needs of application programmers at your site.

After you install Enterprise COBOL, you can:
- Create additional reserved word tables
- Make the compiler options fixed
- Modify the default values for the compiler options
- Modify the residency values for the compiler phases
- Customize Unicode support for COBOL

The following sections discuss the planning you must do to customize Enterprise COBOL for your site.

Planning to modify compiler option default values

Compiler option defaults and residency for compiler phases are set in the IGYCDOPT program, as shown in Table 1 on page 3 and Table 2 on page 9. The default options module, IGYCDOPT, is link-edited with AMODE (31) and RMODE (ANY) during installation.

Planning to Customize Enterprise COBOL

When you assemble COBOL customization parts, such as IGYCDOPT, you need access to a system MACLIB. Typically, the MACLIB is found in SYS1.MACLIB. You also need access to the COBOL MACLIB IGY.V3R2M0.SIGYMAC.

The IGYCDOPT program has two purposes: it lets you select and fix the defaults for compiler options and specify which compiler phases are in shared storage. You can accept the IBM-supplied compiler option values that you receive when you install Enterprise COBOL, or you can modify them to better suit the needs of programmers at your location. You can also choose whether or not your application programmers will have the ability to override these options.

Note: The high-level qualifiers of IGY.V3R2M0 might have been changed when Enterprise COBOL was installed.

If you identify compiler phases that reside in shared system storage, the compiler can use the storage in the region for work areas. For a more detailed description of why you might want to modify the phase defaults, see "Why place the compiler phases in shared storage?" on page 5.

Why make compiler options fixed?

Enterprise COBOL can help you to set up your site's unique programming standards. For example, many sites select RENT as the preferred compiler option, but have no easy way to enforce its use.

With Enterprise COBOL, you can use the IGYCDOPT program to specify that an option is fixed and cannot be changed or overridden at compile time. Then, at compile time, an attempt to override a fixed option is not allowed and results in a diagnostic message with a nonzero compiler return code.

When certain options are fixed for consistent usage, there might be special conditions that require the ability to bypass a fixed option. This change can be made by assembling a temporary copy of the IGYCDOPT program with different parameters. At compile time, if you use a JOBLIB or STEPLIB containing the required IGYCDOPT module, you can bypass the fixed option. For example, if you select the OPT (OPTIMIZE) option to be fixed—indicating that you always want the COBOL compiler to generate optimized object code—and then need to exempt an application from this requirement, you must reassemble the IGYCDOPT program after you remove the asterisk parameter from the option. You then place the resulting IGYCDOPT module in a temporary library to be accessed as a JOBLIB or STEPLIB at compile time.

Sample installation jobs

Enterprise COBOL provides two sample installation jobs that you can modify and then use to change the defaults for compiler options. One sample job provides an example of how to change the IBM-supplied defaults for compilers. The other sample job provides an example of how to override compiler options that have been fixed.

IGYWDOPT
Use this sample installation job to change the IBM-supplied defaults using SMP/E.

IGYWUOPT
Use this sample installation job to create a module outside of SMP/E in which you can specify different defaults if it becomes necessary to override compiler options that have been fixed with the IGYCDOPT program.

These jobs are located in the COBOL sample data set IGY.V3R2M0.SIGYSAMP.

Modifying compiler options and phases

If you plan to modify the values for compiler options and compiler phases, use the IGYCOPT syntax format shown in Figure 1. The IBM-supplied default values are shown both on the planning worksheets and immediately following each syntax diagram. The syntax diagrams also show the default as explained in “How to read the syntax diagrams” on page v.

Compiler options and phases, and their defaults, are discussed in the following sections. Review these options and phases and their default values to determine the values that are most suitable for your applications.

IGYCOPT format

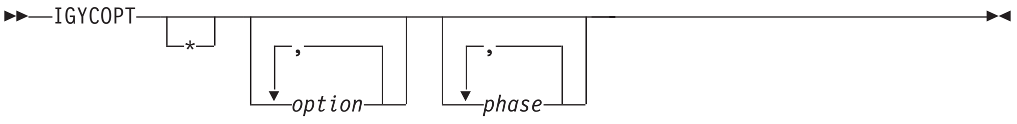


Figure 1. Syntax format for IGYCOPT compiler options and phases macro

IGYCDOPT worksheet for compiler options

The following worksheet will help you to plan and code the compiler options portion of the IGYCDOPT program. To complete the worksheet, fill in the ″Enter * for fixed″ and the ″Enter selection″ columns.

The IGYCDOPT worksheet also includes a section for compiler phases. That section of the worksheet can be found in “IGYCDOPT worksheet for compiler phases” on page 9, following the discussion on compiler phases.

Notes:

1. Coding the asterisk [*], when you modify a compiler option default value, indicates that the option is to be fixed and cannot be overridden by an application programmer.
2. The ALOWCBL, DBCSXREF, LVLINFO, and NUMCLS options cannot be overridden at compile time. Therefore, the ″Enter * for fixed″ worksheet entries for these options are blank.
3. The IBM-supplied default value for ADEXIT, INEXIT, LVLINFO, LIBEXIT, and PRTEXIT is null. Therefore, the ″IBM-supplied default″ entries for these options are blank.
4. The DUMP compiler option cannot be set through the IGYCDOPT program. Unless changed at compile time, DUMP is always set to NODUMP.

Table 1. IGYCDOPT worksheet for options

Compiler option	Enter * for fixed	Enter selection	IBM-supplied default	Syntax description
ADATA=	____	__________	NO	14
ADEXIT=	____	__________		15
ADV=	____	__________	YES	15
ALOWCBL=	____	__________	YES	16
ARITH=	____	__________	COMPAT	16
AWO=	____	__________	NO	17
BUF=	____	__________	4K	17

Table 1. IGYCDOPT worksheet for options (continued)

Compiler option	Enter * for fixed	Enter selection	IBM-supplied default	Syntax description
CICS=	____	__________	**NO**	17
CODEPAGE=	____	__________	**1140**	17
COMPILE=	____	__________	**NOC(S)**	18
CURRENCY=	____	__________	**NO**	19
DATA=	____	__________	**31**	20
DATEPROC=	____	__________	**NO**	21
DBCS=	____	__________	**Yes**	22
DBCSXREF	____	__________	**NO**	22
DECK=	____	__________	**NO**	23
DIAGTRUNC=	____	__________	**NO**	24
DLL=	____	__________	**NO**	24
DYNAM=	____	__________	**NO**	25
EXPORTALL=	____	__________	**NO**	25
FASTSRT=	____	__________	**NO**	27
FLAG=	____	__________	**(I,I)**	26
FLAGSTD=	____	__________	**NO**	26
INTDATE=	____	__________	**ANSI**	29
INEXIT=	____	__________		29
LANGUAGE=	____	__________	**ENGLISH**	30
LIB=	____	__________	**NO**	30
LIBEXIT=	____	__________		31
LINECNT=	____	__________	**60**	31
LIST=	____	__________	**NO**	31
LITCHAR=	____	__________	**QUOTE**	32
LVLINFO=	____	__________		32
MAP=	____	__________	**NO**	33
NAME=	____	__________	**NO**	33
NSYMBOL=	____	__________	**NATIONAL**	34
NUM=	____	__________	**NO**	34
NUMCLS=	____	__________	**PRIM**	35
NUMPROC=	____	__________	**NOPFD**	35
OBJECT=	____	__________	**YES**	36
OFFSET=	____	__________	**NO**	36
OPT=	____	__________	**NO**	37
OUTDD=	____	__________	**SYSOUT**	38
PGMNAME=	____	__________	**COMPAT**	38
PRTEXIT=	____	__________		39
SQL=	____	__________	**NO**	42
RENT=	____	__________	**YES**	39
RMODE=	____	__________	**AUTO**	40
SEQ=	____	__________	**YES**	41
SIZE=	____	__________	**MAX**	41
SOURCE=	____	__________	**YES**	42
SPACE=	____	__________	**1**	42
SSRANGE=	____	__________	**NO**	43
TERM=	____	__________	**NO**	44
TEST=	____	__________	**NO**	44
THREAD=	____	__________	**NO**	44
TRUNC=	____	__________	**STD**	47
VBREF=	____	__________	**NO**	48
WORD=	____	__________	***NO**	48
XREFOPT=	____	__________	**YES**	49

Table 1. IGYCDOPT worksheet for options (continued)

Compiler option	Enter * for fixed	Enter selection	IBM-supplied default	Syntax description
YRWINDOW=	____	__________	**1900**	50
ZWB=	____	__________	**YES**	50

Planning to place compiler phases in shared storage

You might want to make some load modules resident in a link-pack area in order to minimize the search for them when an Enterprise COBOL program is run or when the modules will be shared. You might also want to make some or all of the compiler phases resident. Note that the term *shared storage* is used generally to describe the link-pack area (LPA), the extended link-pack area (ELPA), or modified link-pack area (MLPA). Except where specifically stated, all three terms are referenced when *link-pack area* is used in the following sections.

Why place the compiler phases in shared storage?

All compiler modules, except the run dump modules (IGYCRDPR and IGYCRDSC) and the reserved word utility (IGY8RWTU), are eligible for placement in the shared storage of OS/390 or z/OS machines. All compiler phases except IGYCRCTL and IGYCSIMD have RMODE(ANY) and AMODE(31). Since IGYCRCTL and IGYCSIMD have RMODE 24, they can be placed in the LPA or MLPA, but not in the ELPA. Shared storage is an area of storage that is the same for each virtual address space. Information stored there can be shared and does not have to be loaded in the user region because it is the same space for all users. By sharing the information, more space is made available for the compiler work area.

Note: The Enterprise COBOL, COBOL for MVS & VM, COBOL for OS/390 & VM and VS COBOL II compilers use the same module names; thus, only one set of phases can be placed in the LPA for any given initialization of the operating system.

The IGYCDOPT program indicates where each compiler phase is loaded—either inside (IN) or outside (OUT) of the user region. By placing compiler phases in the MLPA, the compiler has more storage available for the user's program.

If you indicate that a phase will not reside in the user region, you must ensure that you actually place the phase in shared storage. This information is used by the compiler to determine how much storage to leave for the system to load compiler phases in the user region. For a description of how to place a phase in shared storage, see *Initialization and Tuning* manual for your particular operating system, as listed under "Related publications" on page 71.

We recommend that the following four phases be placed in a shared storage area:

IGYCRCTL
because it is resident in the user region throughout compilation

IGYCSIMD
because it is resident in the user region throughout compilation

IGYCPGEN
because it is one of the two largest compiler phases

IGYCSCAN
because it is the other of the two largest compiler phases

You can select any or all compiler phases to be placed in shared storage based on frequency of concurrent use and phase size. If your facility seldom uses the compiler, there might be no advantage to installing any phases in shared storage. However, if there are frequent compilations and sufficient MLPA storage is available, making the entire compiler resident might be advantageous. If sufficient shared storage is not available, priority should be given to IGYCRCTL and IGYCSIMD, the two phases that are always resident in the user region during compilation. Also, if sufficient shared storage is not available, priority should be given to IGYCPGEN and IGYCSCAN, the largest compiler phases.

Another advantage of placing compiler phases in shared storage is that, at compile time, the initialization logic allocates in the user region a storage block of sufficient size to contain the largest phase not resident in shared storage. Minimizing the space allocation for any given user region size means more space for the compilation process (which allows larger programs to be compiled within a given user region) and possibly a more efficient compile. The IGYCPGEN and IGYCSCAN compiler phases are approximately 250-KB larger than the next largest compiler phase. Shared storage can make a significant difference if you are compiling using the 760-KB minimum region size.

Compiler phases and their defaults

To indicate where each compiler phase is loaded in relation to the user region, specify either IN or OUT. See "Why place the compiler phases in shared storage?" on page 5, for more information about why you might or might not want to change these defaults.

IN Indicates that the compiler phase is loaded into the user region from a library available at compile time. The compiler reserves storage for the phase from the value specified in the SIZE option.

Even though IN is specified for a compiler phase, the phase still can be placed into the shared system area. However, the compiler control phase ensures that the main storage area reserved for compiler phases is large enough to contain the largest phase for which IN is specified. This option will cause some storage to be unused.

OUT Indicates that the compiler phase is not loaded into the user region from the library, and therefore must reside in a shared system area, such as the MLPA.

IGYCASM1
The Assembly 1 phase. It determines the object module storage, allocates the permanent and temporary registers, and optimizes addressability for data and procedure references. It also creates object text for data areas.

Syntax

```
►►──ASM1=──┬─IN──┬──────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCASM2
The Assembly 2 phase. It completes preparation of the object program and creates object text, listings, punch data sets, and tables for the debugging feature.

Syntax

```
►►──ASM2=──┬─IN──┬───────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCDIAG

The diagnostic phase. It processes E-form text and generates compiler diagnostics for source program errors. It includes IGYCDIAG plus the following message modules: IGYCxx$D, IGYCxx$1, IGYCxx$2, IGYCxx$3, IGYCxx$4, IGYCxx$5, and IGYCxx$8, where xx is EN, UE, or JA.

Syntax

```
►►──DIAG=──┬─IN──┬───────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCDMAP

The DMAP phase. It prepares text for output requested by the MAP option.

Syntax

```
►►──DMAP=──┬─IN──┬───────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCFGEN

The file generation phase. It generates the control blocks for the FDs and SDs defined in the program.

Syntax

```
►►──FGEN=──┬─IN──┬───────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCINIT

The initialization phase. It does housekeeping to prepare for running of the processing phases.

Syntax

```
►►──INIT=──┬─IN──┬───────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCLIBR

The COPY phase. It processes library source text and does a syntax check of the COPY, BASIS, and REPLACE statements.

Syntax

```
►►──LIBR=──┬─IN──┬───────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCLSTR
The source listing phase. It prints the source listing with embedded cross-reference and diagnostic information.

Syntax

```
►►──LSTR=──┬─IN──┬─────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCMSGT
Represents the header text table and diagnostic message level tables. It includes the following modules: IGYCxx$R, IGYCLVL0, IGYCLVL1, IGYCLVL2, IGYCLVL3, and IGYCLVL8, where xx is EN, UE, or JA.

Syntax

```
►►──MSGT=──┬─IN──┬─────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCOPTM
The optimizer phase. It restructures the PERFORM statements and eliminates duplicate computations.

Syntax

```
►►──OPTM=──┬─IN──┬─────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCOSCN
The option scanning phase. It determines the default options, processes the EXEC PARM options, and processes the PROCESS (CBL) statements.

Syntax

```
►►──OSCN=──┬─IN──┬─────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCPGEN
The procedure generation phase. It supplies code for all procedure source verbs.

Syntax

```
►►──PGEN=──┬─IN──┬─────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCRCTL
The resident control phase. It establishes the size of compiler common and working storage, and performs initialization of program common storage.

Syntax

```
►►──RCTL=──┬─IN──┬─────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCRWT

The normal reserved word table.

Syntax

```
►►──RWT=──┬─IN──┬──────────────────────────────────────────────────►◄
          └─OUT─┘
```

IGYCSCAN

The scanning phase. It does syntax and semantic analysis of the source program and translates the source to intermediate text.

Syntax

```
►►──SCAN=──┬─IN──┬─────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCSIMD

The system interface phase for the Enterprise COBOL compiler. This phase is called by all other compiler phases to perform system-dependent functions.

Syntax

```
►►──SIMD=──┬─IN──┬─────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCXREF

The XREF phase. It sorts user-names and procedure-names in EBCDIC collating sequence.

Syntax

```
►►──XREF=──┬─IN──┬─────────────────────────────────────────────────►◄
           └─OUT─┘
```

IGYCDOPT worksheet for compiler phases

The following worksheet will help you to plan and code the phases portion of the IGYCDOPT program. Circle the value that you plan to assign to each phase. For more information on the values that can be assigned to each phase, see "Compiler phases and their defaults" on page 6.

Note: All phase defaults are initially set to IN.

Table 2. IGYCDOPT program worksheet for compiler phases

Phase	Circle selection	Syntax description
ASM1=	IN / OUT	6
ASM2=	IN / OUT	6
DIAG=	IN / OUT	7
DMAP=	IN / OUT	7
FGEN=	IN / OUT	7
INIT=	IN / OUT	7
LIBR=	IN / OUT	7
LSTR=	IN / OUT	8

Table 2. IGYCDOPT program worksheet for compiler phases (continued)

Phase	Circle selection	Syntax description
MSGT=	**IN** / OUT	8
OPTM=	**IN** / OUT	8
OSCN=	**IN** / OUT	8
PGEN=	**IN** / OUT	8
RCTL=	**IN** / OUT	5
RWT=	**IN** / OUT	9
SCAN=	**IN** / OUT	5
SIMD=	**IN** / OUT	9
XREF=	**IN** / OUT	9

Planning to create an additional reserved word table

When you install Enterprise COBOL, you have access to the following reserved word tables:

- IGYCRWT—The default reserved word table provided for your entire facility.
- IGYCCICS—A CICS-specific reserved word table, provided as an alternate reserved word table (See "CICS reserved word table (IGYCCICS)").

You can create additional reserved word tables after installation. (During compilation, the value of the WORD compiler option determines which reserved word table is used.)

Why create additional reserved word tables?

You can create additional reserved word tables to do the following:

- Translate the reserved words into another language, such as French or German.
- Prevent application programmers from using a particular Enterprise COBOL instruction, such as GO TO.
- Control the usage of nested programs.
- Flag those that are not supported on CICS, such as READ and WRITE.

Controlling use of nested programs

To restrict the use of nested programs, without restricting any other COBOL language features, modify the reserved word table. Do this with the INFO and RSTR control statements. For instructions on how to make these modifications, see "Creating or modifying a reserved word table" on page 56.

Reserved word tables supplied with Enterprise COBOL

Three reserved word tables are on the installation tape:
- Default reserved word table
- CICS reserved word table

Default reserved word table (IGYCRWT)

The default reserved word table is described in an appendix of *Enterprise COBOL Language Reference*.

CICS reserved word table (IGYCCICS)

Enterprise COBOL provides an alternate reserved word table specifically for CICS application programs. It is set up so that COBOL words not supported under CICS are flagged by the compiler with an error message.

The CICS reserved word table is the same as the default reserved word table except that the following COBOL words are marked as restricted (RSTR):

CLOSE	I-O-CONTROL	SD
DELETE	MERGE	SORT
FD	OPEN	START
FILE	READ	WRITE
FILE-CONTROL	RERUN	
INPUT-OUTPUT	REWRITE	

SORT users

If you intend to use the SORT statement under CICS (Enterprise COBOL supports an interface for the SORT statement under CICS), you must modify the CICS reserved word table before using it. The words underlined above must be removed from the list of words marked as restricted, because they are required for the SORT function.

Using the table: To use the CICS reserved word table, you must specify the WORD(CICS) compiler option. To have the CICS reserved word table used as the default, you must set the default value of the WORD compiler option to WORD=CICS.

Location of the table: The data used to create the CICS reserved word table is in member IGY8CICS in IGY.V3R2M0.SIGYSAMP.

Note: The high-level qualifiers of IGY.V3R2M0 might have been changed when Enterprise COBOL was installed.

Chapter 2. Enterprise COBOL compiler options

This describes the compiler options whose default values you can change. The notes that accompany some of the descriptions provide additional information about these options, such as how they interact with other options during compilation. This information might help you to make decisions about which default values are appropriate for your installation. For more information on using the compiler options, see *Enterprise COBOL for z/OS and OS/390 Programming Guide*.

Important

Confer with the application programmers at your site while you plan the customization of Enterprise COBOL. Doing so will ensure that the modifications you make serve their needs and support the applications being developed.

Specifying COBOL compiler options

When you specify compiler options in the IGYCOPT macro, both the option name and its value must be specified in uppercase. If you don't specify the option name in uppercase, both the option name and its value are ignored and the default value is used instead. No error message is issued. If only the option value is not in uppercase, an error message will be issued indicating that an invalid option value has been specified.

Conflicting compiler options

If you specify certain compiler option values, a conflict with other compiler options might result. Table 3 will help you resolve possible conflicts between compiler options.

Table 3. Conflicting compiler options

Compiler option	Conflicts with:

CICS=YES	RENT=NO
	DYNAM=YES
	LIB=NO

DBCS=NO	NSYMBOL=NATIONAL
DBCSXREF=(other than NO)	XREFOPT=NO
DLL=NO	EXPORTALL=YES

DLL=YES	DYNAM=YES
	RENT=NO

DYNAM=YES	CICS=YES
	DLL=YES
	EXPORTALL=YES

EXPORTALL=YES	DLL=NO
	DYNAM=YES
	RENT=NO

LIB=NO	CICS=YES
	SQL=YES

Table 3. Conflicting compiler options (continued)

Compiler option	Conflicts with:
LIST=YES	OFFSET=YES
NSYMBOL=NATIONAL	DBCS=NO
OBJECT=NO	TEST=(other than NO)
OFFSET=YES	LIST=YES
OPT=STD or OPT=FULL	TEST=(other than NONE for hook location)

RENT=NO	CICS=YES
	DLL=YES
	EXPORTALL=YES
	THREAD=YES

SQL=YES	LIB=NO
TEST=(other than NO)	OBJECT=NO
TEST=(other than NONE for hook-location suboption)	OPT=STD or OPT=FULL
THREAD=YES	RENT=NO
WORD=xxxx	FLAGSTD=(other than NO)
XREFOPT=NO	DBCSXREF=(other than NO)

Compiler options for standards conformance

For information on specifying compiler options to conform with the COBOL 85 Standard, please refer to the Enterprise COBOL for z/OS and OS/390 Programming Guide.

Compiler options syntax and descriptions

The syntax diagrams on the following topics describe each modifiable compiler option. The text below each diagram describes the effect of selecting a specific parameter.

Notes:

1. The DUMP option is not in this list. Unless you change it at compile time, DUMP is always set to NODUMP. This option is not for general use; it is used only at the request of an IBM representative.
2. Coding the asterisk [*] , when you modify a compiler option default value, indicates that the option is to be fixed and cannot be overridden by an application programmer.

ADATA

Syntax

```
                        ┌─NO──┐
►►──ADATA=──┬───┬──────┴─YES─┴──────────────────────────────────────────────►◄
            └─*─┘
```

Default
   ADATA=NO

YES
Produces the Associated Data file with the appropriate records.

NO
Does not produce the Associated Data file.

Notes:

1. The ADATA option can be specified only at invocation through the option list, on the PARM field of JCL, as a command option, or as an installation default.
2. Selection of the Japanese language option might result in DBCS characters written records in the Associated Data file.
3. Specification of NOCOMPILE(W|E|S) might stop compilation prematurely, resulting in a loss of specific Associated Data records.
4. Specification of INEXIT prohibits identification of the compilation source file for the Associated Data file.

ADEXIT

Syntax

```
►►──ADEXIT=──┬──────────────┬──────────────────────────────────────────►◄
             └─┬───┬──name──┘
               └─*─┘
```

Default
No exit is specified.

name
Identifies a module to be used with the EXIT compiler option. When the suboption for this user exit is specified, the compiler loads that module and calls it for each record that is written to the Associated Data file. For more information, see Enterprise COBOL for z/OS and OS/390 Programming Guide.

ADV

Syntax

```
                    ┌─YES─┐
►►──ADV=──┬───┬─────┴─NO──┴────────────────────────────────────────────►◄
          └─*─┘
```

Default
ADV=YES

YES
Adds one byte to the record length for the printer control character. This option might be useful to programmers who use WRITE. . .ADVANCING in their source files. The first character of the record does *not* have to be explicitly reserved by the programmer.

NO
Does not adjust the record length for WRITE. . .ADVANCING. The compiler uses the first character of the specified record area to place the printer control character. The application programmer must ensure that the record description allows for this additional byte.

Notes:

1. With ADV=YES, the record length on the physical device is one byte larger than the record description length in the source program.
2. If the record length for the output file is not defined in the source code, COBOL ensures that the DCB parameters are appropriately set.
3. If ADV=YES is specified, and the record length for the output file has been defined in the source code, the programmer *must* specify the record description length as one byte larger than the source program record description. The programmer *must* also specify the block size in correct multiples of the larger record size.
4. If the LINAGE clause is specified in a file description (FD), the compiler treats that file as if ADV=YES has been specified.

ALOWCBL

Syntax

```
                ┌─YES─┐
►►──ALOWCBL=────┴─NO──┴──────────────────────────────────────────────►◄
```

Default
ALOWCBL=YES

YES
Allows the use of the PROCESS (or CBL) statements in COBOL programs.

NO
Diagnoses the use of PROCESS (or CBL) statements in a program as an error.

Notes:

1. ALOWCBL cannot be overridden at compile time because it cannot be included in the PROCESS (or CBL) statement.
2. The PROCESS (or CBL) statement specifies compiler option parameters within source programs. If your installation requirements do not allow compiler options to be specified in a source program, specify ALOWCBL=NO.

ARITH

Syntax

```
                      ┌─COMPAT─┐
►►──ARITH=──┬─────┬───┴─EXTEND─┴─────────────────────────────────────►◄
            └──*──┘
```

Default
ARITH=COMPAT

COMPAT
Specifies 18 digits as the maximum precision for decimal data.

EXTEND
Specifies 31 digits as the maximum precision for decimal data.

AWO

Syntax

```
                ┌─NO──┐
►►──AWO=──┬───┬─┴─YES─┴──────────────────────────────────────────────►◄
          └─*─┘
```

Default
AWO=NO

YES
Activates the APPLY-WRITE-ONLY clause for any file within the program that is physical sequential with variable block format regardless of whether or not the APPLY-WRITE-ONLY clause is specified in the program.

Performance consideration: Using AWO=YES generally results in fewer calls to Data Management Services for run-time files when handling input and output.

NO
Does not activate the APPLY-WRITE-ONLY clause for any file within the program that is physical sequential with variable block format unless the APPLY-WRITE-ONLY clause is specified in the program.

BUF

Syntax

```
                ┌─4K───────┐
►►──BUF=──┬───┬─┼─integer──┼─────────────────────────────────────────►◄
          └─*─┘ └─integerK─┘
```

Default
BUF=4K

integer
Specifies the amount of dynamic storage, in bytes, to be allocated to each compiler work file buffer. The minimum value is 256 bytes.

Performance consideration: Using a large buffer size usually improves the performance of the compiler.

integerK
Specifies the amount of dynamic storage to be allocated to buffers in increments of 1-K (1024) bytes.

Notes:

1. BUF and SIZE values are used by the compiler to determine how much storage to use during compilation. The amount allocated to the buffers is included in the amount of main storage available to the compiler for the SIZE option.
2. BUF cannot exceed the track capacity for the device used, nor can it exceed the maximum allowed by data management services.

CICS

Syntax

```
                    ┌─YES─┐
►►──CICS=──┬───┬────┴─NO──┴──────────────────────────────────────────────►◄
           └─*─┘
```

Default
CICS=NO

NO
When the NO option is specified, any CICS statements that are found in the source program are diagnosed and discarded.

YES
If a COBOL source program contains CICS statements and has not been preprocessed by the CICS translator, the YES option must be specified.

Notes:

1. The CICS compiler option can contain CICS suboptions. The CICS suboptions delimiter can be quotes or apostrophes. CICS suboptions cannot be specified as a COBOL installation default.
2. You can specify the CICS compiler option in any of the compiler option sources: installation defaults, compiler invocation, or PROCESS or CBL statements.

CODEPAGE

Syntax

```
►►──CODEPAGE=──┬───┬──integer────────────────────────────────────────────►◄
               └─*─┘
```

Default
CODEPAGE (1140)

ccsid
ccsid must be a valid coded character set identifier (CCSID) number that identifies an EBCDIC code page.

Notes:

1. The CODEPAGE option specifies the code page used for encoding:
   - Contents of alphanumeric and DBCS data items at run time
   - Alphanumeric, national, and DBCS literals in a COBOL source program
2. The default CCSID 1140 is an equivalent of CCSID 37 (EBCDIC Latin-1, USA) but includes the Euro symbol.

COMPILE

Syntax

```
                            ┌W┐
                     ┌NOC(─┼E┼─)─┐
                     │      └S┘   │
►►──COMPILE=─┬───┬───┴YES─────────┴──────────────────────────────►◄
             └─*─┘
```

Default
COMPILE=NOC(S)

YES
Indicates that you want full compilation, including diagnostics and object code.

NOC
Indicates that you want only a syntax check.

NOC(W)
NOC(E)
NOC(S)
Specifies an error message level: W is warning; E is error; S is severe. When an error of the level specified or of a more severe level occurs, compilation stops, and only syntax checking is done for the balance of the compilation.

Note: Specifying NOCOMPILE might affect the Associated Data file by stopping compilation prematurely, resulting in loss of specific messages.

CURRENCY

Syntax

```
                      ┌NO──────┐
►►──CURRENCY=─┬───┬───┴literal─┴─────────────────────────────────►◄
              └─*─┘
```

The COBOL default currency symbol is the dollar sign ($). The CURRENCY option allows you to define an alternate default currency symbol.

Default
CURRENCY=NO

literal
Represents the default currency symbol that you want to use in your program.

The literal must be a nonnumeric literal representing a one-byte EBCDIC character that must not be any of the following:

- Digits zero (0) through nine (9)
- Uppercase alphabetic characters: A B C D P R S V X Z
- Lowercase alphabetic characters a through z
- The space
- Special characters: * + - / , . ; ( ) = ″
- Uppercase alphabetic character G, if the COBOL program defines a DBCS item with the PICTURE symbol G. The PICTURE clause will not be valid for that DBCS item because the symbol G is considered to be a currency symbol in the PICTURE clause.

- Uppercase alphabetic character N, if the COBOL program defines a DBCS item with the PICTURE symbol N. The PICTURE clause will not be valid for that DBCS item because the symbol N is considered to be a currency symbol in the PICTURE clause.
- Uppercase alphabetic character E, if the COBOL program defines an external floating-point item. The PICTURE clause will not be valid for the external floating-point item because the symbol E is considered to be a currency symbol in the PICTURE clause.

The literal (including hex literal) syntax rules are as follows:
- The literal delimiters can be either quotes or apostrophes regardless of any option setting for literal delimiters.
- When an apostrophe (’) is to be the currency sign, the embedded apostrophe must be doubled, that is, two apostrophes must be coded to represent one apostrophe within the literal. For example:
  ```
  '''' or "''"
  ```
- The format for a hex literal specification is as follows:
  ```
  X'H1H2' or X"H1H2"
  ```

  where H1H2 is a valid hexadecimal value representing a one-byte EBCDIC character conforming to the rules for the currency sign literal as described above. Alphabetic characters in the hex literal must be in uppercase.

  Note: Hex values of X’20’ or X’21’ are not allowed.

NO
Indicates that no alternate default currency sign is provided through the CURRENCY option, and the dollar sign will be used as the default currency sign for the program if the CURRENCY option is not specified at compile time.

The value NO provides the same results for the source program as omitting the CURRENCY SIGN clause in the COBOL source program.

Notes:
1. You can use the CURRENCY option as an alternative to the CURRENCY SIGN clause (which is specified in the COBOL source program) for selecting the currency symbol that you use in the PICTURE clause of your COBOL program.
2. When both the CURRENCY option and the CURRENCY SIGN clause are used in a program, the symbol that is specified in the CURRENCY SIGN clause is the currency symbol in a PICTURE clause when that symbol is used (even if the CURRENCY option is fixed {*}).

DATA

Syntax

```
                      ┌─31─┐
►►──DATA=──┬───┬──────┴─24─┴──────────────────────────────────────────►◄
           └─*─┘
```

Default
DATA=31

31 Causes allocation of user data areas, such as working storage and FD record areas, from unrestricted storage or in space acquired by a GETMAIN with the

LOC=ANY option. Specifying this option can result in storage being acquired in virtual addresses either above or below the 16-MB line. The operating system generally satisfies the request with space in virtual addresses above the 16-MB line, if it is available.

24 Causes allocation of user data areas in virtual addresses below 16 megabytes in storage acquired by a GETMAIN with the LOC=BELOW option.

Specify DATA=24 for programs compiled with the RENT option that are passing data parameters to programs in 24-bit mode. This includes the following cases:

- An Enterprise COBOL program is passing items in its WORKING-STORAGE to an AMODE 24 program.
- An Enterprise COBOL program is passing, by reference, data items received from its caller to an AMODE 24 program. DATA=24 is required even when the data received is below the 16-MB line.

Otherwise, the data might not be addressable by the called program.

Notes:

1. When a program is compiled with the RENT option, the DATA option controls how space for WORKING-STORAGE and parameter lists is acquired.
2. The DATA option has no effect on programs compiled with the NORENT option.

DATEPROC

Syntax

```
                        ┌─NO──────────────────────────────────────┐
►►──DATEPROC=─┬───┬─────┴─(─┬─FLAG───┬─┬─────────────────┬─)──────┴──────────────────────►◄
              └─*─┘         └─NOFLAG─┘ └─,─┬─TRIG────┬───┘
                                           └─NOTRIG──┘
```

The DATEPROC option determines whether the compiler will perform date processing using the DATE FORMAT clause and other language constructs.

Default
DATEPROC=NO, or DATEPROC(FLAG,NOTRIG) if only DATEPROC is specified

FLAG
Recognizes the DATE FORMAT clause and performs automatic date processing. In addition, specifying DATEPROC=FLAG flags, either with an information-level message or a warning-level message as appropriate, each statement that uses or is affected by date processing.

NOFLAG
Recognizes the DATE FORMAT clause and performs automatic date processing. Statements that use or are affected by date processing are not flagged with information-level or warning-level messages.

TRIG
Recognizes the DATE FORMAT clause and performs date processing based on automatic windowing that the compiler applies to operations on windowed date fields. The automatic windowed date fields are sensitive to specific trigger

or limit values in the date fields and in other nondate fields. These specific values represent dates that are not valid and that can be tested for or used as upper or lower limits.

NOTRIG
Recognizes the DATE FORMAT clause and performs date processing based on automatic windowing that the compiler applies to operations on windowed date fields. The automatic windowed date fields are not sensitive to specific trigger or limit values in the date fields and in other nondate fields. Only the value of the year part of dates is relevant to automatic windowing.

NO
Treats the DATE FORMAT clause as comments and disables automatic date processing. In the case of the new intrinsic functions, specifying DATEPROC=NO generates object code that returns a default value whenever a new intrinsic function is used.

Note:

Error-level and severe-level messages are issued regardless of whether DATEPROC=FLAG or DATEPROC=NOFLAG is specified.

DBCS

Syntax

```
                     ┌─YES─┐
►►──DBCS=──┬───┬─────┴─NO──┴──────────────────────────────────────►◄
           └─*─┘
```

Default
DBCS=YES

YES
Recognizes X'0E' and X'0F' in a nonnumeric literal and treats them as shift-out and shift-in control characters for delimiting DBCS data.

NO
Does not recognize X'0E' and X'0F'as shift-out and shift-in control characters in a nonnumeric literal.

Notes:

1. The presence of DBCS data inside the nonnumeric literal might cause the compiler to disallow certain uses of that literal. For example, DBCS characters are not allowed as program names or DDNAMES.
2. DBCS=NO conflicts with NSYMBOL(NATIONAL)

DBCSXREF

Syntax

```
               ┌─NO─────────────────────────────┐
►►──DBCSXREF=──┴─(──┬─R─┬─,xx──┬──────────────┬─)─┴───────────────►◄
                    └─N─┘      └─,yy─┬─────┬──┘
                                     └─,zz─┘
```

Default
DBCSXREF=NO

NO
Specifies that no ordering program is used for cross-reference of DBCS names. If the XREF phase is specified, a cross-reference listing of DBCS names is provided based on their physical order in the program.

R Specifies that the DBCS Ordering Support Program (DBCSOS) is loaded into the user region.

N Specifies that the DBCS Ordering Support Program (DBCSOS) is loaded into a shared system area such as the MLPA.

xx Names a load module of the relevant ordering program to produce DBCS cross references. It must be eight characters in length.

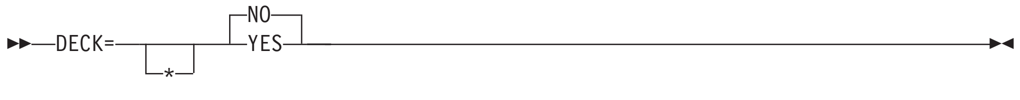
yy Names an ordering type. It must be two characters in length. The default ordering type defined by the specified ordering program occurs if this parameter is omitted.

zz Names the encode table that the specified ordering type uses. It must be eight characters in length. The default encode table that is associated with the particular ordering type occurs if this parameter is omitted.

Notes:
1. The DBCS Ordering Support Program (DBCSOS) must be installed to specify anything other than DBCSXREF=NO.
2. If R is specified and the SIZE value is anything other than MAX, ensure that the user region is large enough to accommodate both the compiler and ordering program.
3. Specifying both XREFOPT=NO and DBCSXREF with an ordering program results in a nonzero return code while attempting to assemble the customization macro.
4. The assembly process terminates when validation diagnoses:
   - A parameter length that is not valid
   - Characters other than 'R' and 'N'
   - Missing parameters after a comma
   - Missing 'yy' when 'zz' is specified

DECK

Syntax



Default
DECK=NO

YES
Places the generated object code in a file defined by SYSPUNCH.

NO
Sends no object code to SYSPUNCH.

DIAGTRUNC

Syntax

```
                          ┌─NO──┐
►►──DIAGTRUNC=──┬───┬─────┴─YES─┴──────────────────────────────────────────►◄
                └─*─┘
```

Default
DIAGTRUNC=NO

YES
Causes the compiler to issue a severity-4 (warning) diagnostic message for MOVE statements with numeric receivers when the receiving data item has fewer integer positions than the sending data item or literal.

NO
Does not cause the compiler to produce a severity-4 message.

Notes:

1. The diagnostic is also issued for moves to numeric receivers from alphanumeric data names or literal senders, except when the sending field is reference modified.
2. There is no diagnostic for COMP-5 receivers, nor for binary receivers when you specify the TRUNC(BIN) option.

DLL

Syntax

```
                    ┌─NO──┐
►►──DLL=──┬───┬─────┴─YES─┴────────────────────────────────────────────────►◄
          └─*─┘
```

Default
DLL=NO

YES
Generates an object module that is enabled for dynamic link library (DLL) support. DLL enablement is required if the program is part of a DLL, references DLLs, or contains object-oriented COBOL syntax (for example, INVOKE statements, or class definitions).

Specification of the DLL option requires that the NODYNAM option and RENT options are also used.

NO
Generates an object module that is not enabled for DLL usage.

DYNAM

Syntax

```
                  ┌─NO──┐
►►──DYNAM=──┬───┬─┴─YES─┴──────────────────────────────────────────────►◄
            └─*─┘
```

Default
DYNAM=NO

YES
Dynamically loads subprograms that are invoked through the CALL literal statement.

Performance consideration: Using DYNAM=YES eases subprogram maintenance because the application is not relink-edited if the subprogram is changed. However, individual applications with CALL literal statements can experience some performance degradation due to a longer path length.

NO
Includes, in the calling program, the text files of subprograms called with a CALL literal statement into a single module file.

Notes:

1. The DYNAM option has no effect on the CALL identifier statement at compile time. The CALL identifier statement always compiles to a dynamic call.
2. Do not specify DYNAM=YES for applications running under CICS.

EXPORTALL

Syntax

```
                      ┌─NO──┐
►►──EXPORTALL=──┬───┬─┴─YES─┴──────────────────────────────────────────►◄
                └─*─┘
```

Default
EXPORTALL=NO

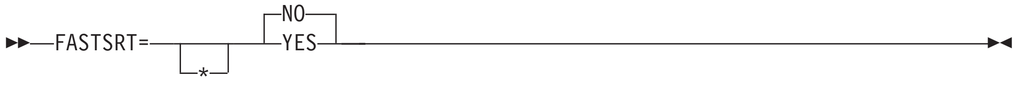
YES
Automatically exports certain symbols when the object deck is link-edited to form a DLL.

Specification of EXPORTALL requires that the DLL, RENT, and NODYNAM options are also used.

NO
Does not export any symbols.

FASTSRT

Syntax



Default
FASTSRT=NO

YES
Specifies that the IBM DFSORT™ licensed program or comparable product performs input and output when you use either the USING or GIVING option.

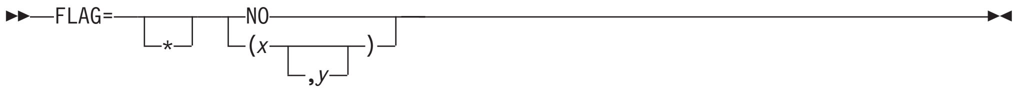
Performance consideration: Using FASTSRT=YES eliminates the overhead, in terms of CPU time usage, of returning to Enterprise COBOL after each record is processed. However, there are restrictions that you must follow if you choose to use this option. (For a detailed description of the restrictions, see *Enterprise COBOL for z/OS and OS/390 Programming Guide*.)

NO
Specifies that Enterprise COBOL does the input and output for the sort and merge.

Notes:
1. If FASTSRT is in effect at compile time, the compiler verifies that the FASTSRT interface can be used for all restrictions except these two:
   - A device other than a direct-access device must be used for sort work files.
   - The DCB parameter of the DD statement for the input file or output file must match the file description (FD) of the file.
2. If FASTSRT cannot be used, the compiler generates a diagnostic message and prevents the sort program from performing I/O when using either the USING or GIVING options. Therefore, it might be to your advantage to specify YES as the default.

FLAG

Syntax



Default
FLAG=(I,I)

Note: The second severity level used in this syntax must be equal to or higher than the first.

x I | W | E | S | U

Specifies that errors at or above the severity level specified are flagged and written at the end of the source listing.

ID	Type	Return Code
I	Information	0
W	Warning	4
E	Error	8
S	Severe error	12
U	Unrecoverable error	16

y I|W|E|S|U

The optional second severity level specifies the level of syntax messages embedded in the source listing in addition to being at the end of the listing.

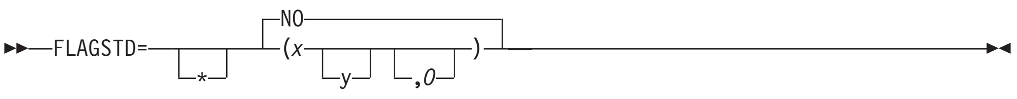
NO
Indicates that no error messages are flagged.

Notes:

1. If the messages are to be embedded, SOURCE must be specified at compile time. Embedded messages enhance productivity because they are placed after the referenced source statement.
2. Specification of FLAG(W|E|S) might result in the loss of entire classes of messages from the Events records in the Associated Date file. See *Enterprise COBOL for z/OS and OS/390 Programming Guide* for more information.

FLAGSTD

Syntax



Default
FLAGSTD=NO

x Can be M, I, or H to specify flagging for a FIPS COBOL subset or standard.

M = ANS minimum subset of Standard COBOL.

I = ANS intermediate subset, composed of those additional intermediate subset language elements that are not part of the ANS minimum subset.

H = ANS high subset, composed of those additional high subset language elements that are not part of the ANS intermediate subset.

y Can be any one or two combinations of D, N, or S to further define the level of flagging produced.

D Specifies ANS Debug module Level 1

N Specifies ANS Segmentation Module Level 1

S Specifies ANS Segmentation Module Level 2 (S is a superset of N.)

O Specifies that obsolete elements occurring in any of the above sets are flagged.

NO
Specifies that no FIPS flagging is accomplished.

Enterprise COBOL compiler options

Notes:

1. The following elements are flagged as nonconforming nonstandard IBM extensions to the COBOL 85 Standard:
   - Language syntax, such as the DATE FORMAT clause, used by the COBOL automatic date-processing facilities (as enabled by the DATEPROC compiler option)
   - Language syntax for object orientation and improved interoperability with C/C++
   - Use of the PGMNAME=LONGMIXED compiler option
2. When FIPS flagging is specified, informational messages in the source program listing identify:
   - Whether the language element is obsolete, nonconforming standard, or nonconforming nonstandard (language elements that are both obsolete and nonconforming are flagged as obsolete only)
   - The clause, statement, or header containing the nonconforming or obsolete syntax
   - The source program line and an indication of the starting column within that line
   - The level or optional module to which the language element belongs
3. FIPS flagging is suppressed when any error diagnosed as level E or higher occurs.
4. Interaction of FLAGSTD and other compiler options:
   - If the following compiler options are explicitly or implicitly specified in a program, FLAGSTD=(other than NO) issues a warning message during compile time:

ADV=NO	LITCHAR=APOST
DATEPROC=(other than NO)	NUM=YES
DBCS=YES	NUMPROC=PFD
DYNAM=NO	SEQ=YES
FASTSRT=YES	TRUNC=OPT or BIN
LIB=NO	WORD=(other than NO or RWT)
	ZWB=NO

   - Specifying the following options together with FLAGSTD=(other than NO), while attempting to assemble the customization macro, results in a nonzero return code.

ADV=NO	LITCHAR=APOST
DATEPROC=(other than NO)	NUM=YES
DBCS=YES	NUMPROC=PFD
DYNAM=NO	SEQ=YES
LIB=NO	TRUNC=OPT or BIN
	WORD=(other than NO or RWT)
	ZWB=NO

5. FLAGSTD might produce events records in the Associated Data file for FIPS standard conformation messages. Error messages are not guaranteed to be sequential in regard to source record numbers.

INEXIT

Syntax

```
►►──INEXIT=──┬───┬──┬──────┬──────────────────────────────────────────────►◄
             └─*─┘  └─name─┘
```

Default
No exit is specified.

name
Identifies a module to be used with the EXIT compiler option. When the suboption for this user exit is specified, the compiler loads that module and calls it to obtain source statements instead of reading the SYSIN data set. When the option is supplied, the SYSIN data set is not opened. For more specific information, see *Enterprise COBOL for z/OS and OS/390 Programming Guide*.

Note:

Specification of INEXIT prohibits identification of the compilation source file.

INTDATE

Syntax

```
                          ┌─ANSI───┐
►►──INTDATE=──┬───┬──┴─LILIAN─┴──────────────────────────────────────►◄
              └─*─┘
```

Default
INTDATE=ANSI

ANSI
Uses the ANSI COBOL Standard starting date for integer date format dates used with date intrinsic functions. Day 1 = Jan 1, 1601.

With INTDATE(ANSI), the date intrinsic functions return the same results as in COBOL/370™ Release 1.

LILIAN
Uses the Language Environment Lilian starting date for integer date format dates used with date intrinsic functions. Day 1 = Oct 15, 1582.

With INTDATE(LILIAN), the date intrinsic functions return results compatible with the Language Environment date callable services. These results are different from those in COBOL/370 Release 1.

Notes:
1. When INTDATE(LILIAN) is in effect, CEECBLDY is not usable because you have no way to turn an ANSI integer into a meaningful date using either intrinsic functions or callable services. If you code a CALL literal statement with CEECBLDY as the target of the call with INTDATE(LILIAN) in effect, the compiler diagnoses this and converts the call target to CEEDAYS.
2. If you set your installation option to INTDATE(LILIAN), you should recompile all of your COBOL/370 Release 1 programs 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, pass them between programs, and even pass them from PL/I to COBOL to C programs and have no problems.

LANGUAGE

Syntax

```
►►──LANGUAGE=──┬───┬──XX──────────────────────────────────────────►◄
               └─*─┘
```

Default
LANGUAGE=EN

XX
Specifies the language for compiler output messages. Entries for this parameter might be selected from the following list.

Table 4. Entries for the LANGUAGE compiler option

Entry	Language
EN or **ENGLISH**	Mixed case U.S. English
JA, **JP**, or **JAPANESE**	Japanese
UE or **UENGLISH**	Uppercase U.S. English

Notes:

1. The LANGUAGE option name must consist of at least the first two identifying characters. Other characters following the first two identifiers can be used; however, only the first two are used to determine the language name.
2. This compiler option does not affect the language in which run-time messages are displayed. For more information on run-time options and messages, refer to *IBM Language Environment for OS/390 & VM Programming Guide*.
3. Some printers use only uppercase and might not accept output in mixed case (LANGUAGE=ENGLISH).
4. To specify the Japanese language option, the Japanese National Language Feature must be installed.
5. To specify the English language option (mixed-case English), the U.S. English Language Feature must be installed.
6. If your installation provides a language other than those listed above, and you select it as your installation's default, you must specify at least the first two characters of the language name. These two characters must be alphanumeric.
7. The selection of Japanese, together with specification of the EVENTS option or the ADATA option, might result in DBCS characters being written to error identification records in the Associated Data file.

LIB

Syntax

```
                      ┌─NO──┐
►►──LIB=──┬───┬───────┴─YES─┴─────────────────────────────────────►◄
          └─*─┘
```

Default
LIB=NO

YES
Indicates that the source program contains COPY or BASIS statements.

NO
Indicates that the source program doesn’t contain COPY or BASIS statements.

LIBEXIT

Syntax

```
►►──LIBEXIT=──┬───┬──┬──────┬───────────────────────────────────────►◄
              └─*─┘  └─name─┘
```

Default
No exit is specified.

name
Identifies a module used with the EXIT compiler option. When the suboption for this user exit is specified, the compiler loads that module and calls it to obtain COPY statements instead of reading the SYSLIB or library-name data set. When the option is supplied, the SYSLIB and library-name data sets are not opened. For more specific information, see *Enterprise COBOL for z/OS and OS/390 Programming Guide*.

LINECNT

Syntax

```
                        ┌─60──────┐
►►──LINECNT=──┬───┬──┴─integer─┴──────────────────────────────────►◄
              └─*─┘
```

Default
LINECNT=60

integer
Specifies the number of lines to be printed on each page of the compiler source code listing. Three of the lines are used to generate headings. For example, if you specify LINECNT=60, 57 lines of source code are printed on each page of the output listing, and 3 lines are used for headings.

Note: The LINECNT installation option is equivalent to the LINECOUNT compile-time option.

LIST

Syntax

```
                     ┌─NO──┐
►►──LIST=──┬───┬──┴─YES─┴─────────────────────────────────────────►◄
           └─*─┘
```

Default
LIST=NO

YES
Produces a listing that includes:
- The assembler-language expansion of source code
- Information about working storage
- Global tables
- Literal pools

NO
Suppresses this listing.

Note: The LIST and OFFSET compiler options are mutually exclusive. Setting OFFSET=YES and LIST=YES results in a nonzero return code and an error message during assembly of the customization macro.

LITCHAR

Syntax

```
                        ┌QUOTE┐
►►──LITCHAR=──┬───┬──┴APOST┴──────────────────────────────────────►◄
              └─*─┘
```

Default
LITCHAR=QUOTE

QUOTE
Use QUOTE if you want the figurative constant [ALL] QUOTE or [ALL] QUOTES to represent one or more quotation mark (") characters. QUOTE conforms to the COBOL 85 Standard.

APOST
Use APOST if you want the figurative constant [ALL] QUOTE or [ALL] QUOTES to represent one or more apostrophe (′) characters.

Notes:
1. Either quotes or apostrophes can be used as literal delimeters, regardless of whether the APOST or QUOTE option is in effect.
2. The delimiter character used as the opening delimiter for a literal must be used as the closing delimiter for that literal.

LVLINFO

Syntax

```
►►──LVLINFO=──┬──────┬────────────────────────────────────────────►◄
              └─xxxx─┘
```

Default
No characters are specified.

xxxx
Identifies the one to four alphanumeric characters that are inserted into the

listing header following the release number (the last 4 bytes of the signature area). This option might be used to identify "compiler level" information within the listing header.

MAP

Syntax

```
                      ┌─NO──┐
►►──MAP=──┬───┬───────┴─YES─┴──────────────────────────────────────────►◄
          └─*─┘
```

Default
MAP=NO

YES
Maps items declared in the DATA DIVISION. Map output includes:
- DATA DIVISION map
- Global tables
- Literal pools
- Program statistics
- Size of the program's working storage and its location in the object code if the program is compiled without the RENT compiler option

NO
Mapping is not performed.

NAME

Syntax

```
                        ┌─NO──────┐
►►──NAME=──┬───┬────────┼─NOALIAS─┼────────────────────────────────────►◄
           └─*─┘        └─ALIAS───┘
```

Default
NAME=NO

NOALIAS
Appends a linkage editor NAME statement (NAME *modname*(R)) to each object module created in a batch compilation. The module name (*modname*) is derived from the PROGRAM-ID statement according to the rules for forming external module names.

ALIAS
Precedes the NAME statement corresponding to the PROGRAM-ID with a linkage editor ALIAS statement for each ENTRY statement in the program.

NO
Does not append linkage editor NAME statements.

Notes:
1. The NAME option allows you to create multiple modules in a program library with a single batch compilation. This can be useful for dynamic calls.

NSYMBOL

Syntax

```
                         ┌─NATIONAL─┐
►►──NSYMBOL=──┬───┬──────┴─DBCS─────┴────────────────────────────────────►◄
              └─*─┘
```

Default
NSYMBOL=NATIONAL

DBCS
Use DBCS when data items are defined with the PICTURE clause consisting only of the PICTURE symbol N and without the USAGE clause. Such data items are treated as if the USAGE DISPLAY-1 clause were specified. Literals of the form N″. . .″ or N′. . .′ are treated as DBCS literals.

NATIONAL
Use NATIONAL when data items are defined with the PICTURE clause consisting only of the PICTURE symbol N and without the USAGE clause. Such data items are treated as if the USAGE NATIONAL clause were specified. Literals of the form N″. . .″ or N′. . .′ are treated as national literals.

Notes:

1. The NSYMBOL(DBCS) option is compatible with previous releases of IBM COBOL. The NSYMBOL(NATIONAL) option handles the N symbol consistently with the 200x COBOL standard.
2. NSYMBOL(NATIONAL) forces the DBCS option.

NUM

Syntax

```
                    ┌─NO──┐
►►──NUM=──┬───┬─────┴─YES─┴──────────────────────────────────────────────►◄
          └─*─┘
```

Default
NUM=NO

YES
Uses the line numbers from the source program rather than compiler-generated line numbers for error messages and procedure maps.

NO
Uses the compiler-generated line numbers for error messages and procedure maps.

Notes:

1. If COBOL programmers use COPY statements and NUM=YES is in effect, they must ensure that the source program line numbers and the COPY member line numbers are coordinated.

NUMCLS

Syntax

```
              ┌─PRIM─┐
►►──NUMCLS=───┴─ALT──┴──────────────────────────────────────────────►◄
```

Default
NUMCLS=PRIM

ALT
Specifies the sign representations that are recognized as valid by the numeric class test for data items that are defined:
- As signed (with an "S" in the PICTURE clause)
- Using DISPLAY or COMPUTATIONAL-3 (packed-decimal)
- Without the SEPARATE phrase on any SIGN clause

Processing with ALT accepts hexadecimal A through F as valid.

PRIM
Processing with PRIM accepts hexadecimal C, D, and F as valid.

Notes:
1. The numeric class test is affected by how both the NUMPROC and the NUMCLS options are specified.
2. The NUMCLS option is effective only for NUMPROC=MIG or NUMPROC=NOPFD. NUMPROC=PFD specifies more strict rules for valid sign configuration.

NUMPROC

Syntax

```
                          ┌─NOPFD─┐
►►──NUMPROC=──┬─────┬─────┼─MIG───┼─────────────────────────────────►◄
              └─*───┘     └─PFD───┘
```

Default
NUMPROC=NOPFD

MIG
Aids in migrating OS/VS COBOL application programs to Enterprise COBOL. Processing with MIG entails these actions:
- Using existing signs for comparison and arithmetic operations
- Generating preferred signs for the results of MOVE and arithmetic operations (These results meet the criteria for using NUMPROC=PFD.)
- Performing numeric rather than logical comparisons

NOPFD
Repairs signs on input. After repair is performed, the signs meet the criteria for NUMPROC=PFD.

PFD
Optimizes the generated code, especially when OPT=STD or OPT=FULL. No explicit sign repair is performed. Note that NUMPROC=PFD has stringent criteria to produce correct results. To use NUMPROC=PFD:

- The sign position of unsigned numeric items must be X'F'.
- The sign position of signed numeric items must be either X'C' if positive or zero, or must be X'D' if negative.
- The sign position of separately signed numeric items must be either '+' if positive or zero, or '-' if otherwise.

Elementary MOVE and arithmetic statements in Enterprise COBOL always generate results with these preferred signs; however, group MOVEs and redefinitions might produce nonconforming results. The numeric class test can be used for verification. With NUMPROC=PFD, a numeric item fails the numeric class test if the signs do not meet the preferred sign criteria.

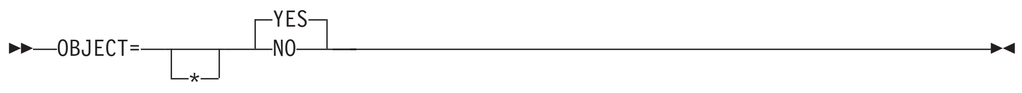
Performance consideration: Using NUMPROC=PFD generates significantly more efficient code for numeric comparisons. For most references to COMP-3 and DISPLAY numeric data items, using NUMPROC=MIG and NUMPROC=NOPFD generates extra code because of sign “fix-up” processing. This extra code might also inhibit some other types of optimizations. Before setting this option, consult with your application programmers to determine the effect on the application program’s output.

Notes:

1. Both the NUMPROC and NUMCLS options affect the numeric class test. With NUMPROC=MIG or NUMPROC=NOPFD, the results of the numeric class test are controlled by how NUMCLS is set. When NUMPROC=PFD, a data item must meet the preferred sign criteria to be considered numeric.

OBJECT

Syntax



Default
OBJECT=YES

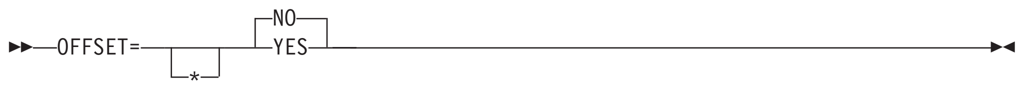
YES
Places the generated object code in a file defined by SYSLIN.

NO
Places no object code in SYSLIN.

Note: The OBJECT=NO option conflicts with all values for TEST other then NO.

OFFSET

Syntax



Default
OFFSET=NO

YES
Produces a condensed PROCEDURE DIVISION listing. The procedure portion of the listing will contain line numbers, verb references, and the location of the first instruction generated for each verb. In addition, the following are produced:

- Global tables
- Literal pools
- Program statistics
- Size of the program's working storage, and its location in the object code if the program is compiled with the NORENT compiler option

NO
Does not condense the listing or produce the items listed above.

Note: The LIST and OFFSET compiler options are mutually exclusive. Setting OFFSET=YES and LIST=YES results in a nonzero return code when attempting to assemble the customization macro. See "Conflicting compiler options" on page 13 for more information on conflict resolution.

OPTIMIZE

Syntax

```
                    ┌NO──┐
►►──OPT=──┬───┬────┼STD─┼──────────────────────────────────────►◄
          └─*─┘    └FULL┘
```

Default
OPT=NO

NO
Specifies that your code is not optimized.

STD
Generates optimized object code.

Performance consideration: Using OPT=STD or OPT=FULL generally results in more efficient run-time code.

FULL
Discards any unused data items and does not generate code for any VALUE clauses for these data items. If the OPT(FULL) and MAP option are both specified, discarded data items will have a BL number of XXXX in the MAP output indicating that the number is not used.

Notes:

1. The OPTIMIZE compiler option is now fully supported for programs containing object-oriented syntax for Java interoperability.
2. If you want to debug optimized object code using Debug Tool, the only hook-location subparameter allowed for the TEST option is NONE. Any other combination results in a nonzero return code and an error message during installation.
3. Optimization is turned off if an S-level error or higher occurs.

OUTDD

Syntax

```
                       ┌SYSOUT─┐
►►──OUTDD=──┬───┬──────┴ddname─┴─────────────────────────────────────►◄
            └─*─┘
```

Default
OUTDD=SYSOUT

ddname
Specifies the ddname of the file used for run-time DISPLAY output.

Notes:

1. See *Language Environment Programming Reference* description of the MSGFILE run-time option to see how OUTDD interacts with MSGFILE.
2. Change the default for this option if, at run time, you expect to have a conflict with another product that requires SYSOUT as a ddname.

PGMNAME

Syntax

```
                          ┌COMPAT────┐
►►──PGMNAME=──┬───┬───────┼LONGMIXED─┼───────────────────────────────►◄
              └─*─┘       └LONGUPPER─┘
```

Default
PGMNAME=COMPAT

LONGMIXED
Program names are processed as is, without truncation, translation, or folding to uppercase.

LONGUPPER
Program names are folded to uppercase by the compiler but otherwise are processed as is, without truncation or translation.

COMPAT
Program names are processed in a manner compatible with COBOL/370 Release 1, and VS COBOL II.

Notes:

1. The PGMNAME option controls the handling of names used in the following contexts:
   - Program names defined in the PROGRAM-ID paragraph
   - Program entry point names on the ENTRY statement
   - Program name references in calls to nested programs and static calls to separately compiled programs
   - Program name references in the static SET procedure-pointer TO ENTRY literal statement
   - Program name references in CANCEL of a nested program
2. Class and method names are not affected by the PGMNAME option.

3. For more specific information, see *Enterprise COBOL for z/OS and OS/390 Programming Guide*.

PRTEXIT

Syntax

```
►►──PRTEXIT=──┬───┬──┬──────┬──────────────────────────────►◄
              └─*─┘  └─name─┘
```

Default
No exit is specified.

name
Identifies a module to be used with the EXIT compiler option. When the suboption for this user exit is specified, the compiler loads that module and calls it instead of writing to the SYSPRINT data set. When the option is supplied, the SYSPRINT data set is not opened. For more specific information, see the *Enterprise COBOL for z/OS and OS/390 Programming Guide*.

RENT

Syntax

```
                     ┌─YES─┐
►►──RENT=──┬───┬──┴─NO──┴──────────────────────────────►◄
           └─*─┘
```

Default
RENT=YES

YES
Indicates that the object code produced for a COBOL program is reentrant. Using RENT=YES enables the program to be placed in shared storage for running above the 16-MB line. However, this option causes the compiler to generate additional code to ensure that the application program is reentrant.

NO
Indicates that the object code produced for a COBOL program is to be nonreentrant.

Notes:

1. Programs must be compiled with RENT=YES or RMODE(ANY), if they will be run in virtual storage addresses above 16-MB.
2. The RENT compiler option is required for programs that are run on CICS.
3. Programs compiled with Enterprise COBOL always have AMODE(ANY). The RMODE assigned to a program depends on the RENT/NORENT and RMODE compiler options. Valid combinations include:

Table 5. Effect of RENT and RMODE on residency mode

RENT/NORENT setting	RMODE setting	Residency mode assigned
RENT	AUTO	RMODE ANY
RENT	ANY	RMODE ANY

Table 5. Effect of RENT and RMODE on residency mode (continued)

RENT/NORENT setting	RMODE setting	Residency mode assigned
RENT	24	RMODE 24
NORENT	AUTO	RMODE 24
NORENT	ANY	RMODE ANY
NORENT	24	RMODE 24

4. When the THREAD compiler option is specified, the RENT compiler option must also be specified. If THREAD and NORENT are specified at the same level of precedence, the RENT option is forced on.

RMODE

Syntax

```
                        ┌─AUTO─┐
►►──RMODE=──┬───┬──────┼─24───┼──────────────────────────────────────►◄
            └─*─┘      └─ANY──┘
```

Default
RMODE=AUTO

AUTO
Specifies that a program will have RMODE 24 if NORENT is specified, and RMODE ANY if RENT is specified.

24 Specifies that a program will have RMODE 24 whether NORENT or RENT is specified.

ANY
Specifies that a program will have RMODE ANY whether NORENT or RENT is specified.

Notes:

1. Enterprise COBOL NORENT programs that pass data to programs running in AMODE 24 must be either compiled with the RMODE (24) option or link-edited with RMODE 24. The data areas for NORENT programs will be above the 16-MB line or below the 16-MB line depending on the RMODE of the program, even if DATA(24) has been specified. DATA(24) applies to programs compiled with the RENT option only.
2. Programs compiled with Enterprise COBOL always have AMODE ANY. The RMODE assigned to a program depends on the RMODE and RENT/NORENT compiler options. Valid combinations include:

Table 6. Effect of RMODE and RENT/NORENT on residency mode

RMODE setting	RENT/NORENT setting	Residency mode assigned
AUTO	RENT	RMODE ANY
AUTO	NORENT	RMODE 24
ANY	RENT	RMODE ANY
ANY	NORENT	RMODE ANY
24	RENT	RMODE 24

Table 6. Effect of RMODE and RENT/NORENT on residency mode (continued)

RMODE setting	RENT/NORENT setting	Residency mode assigned
24	NORENT	RMODE 24

SEQ

Syntax

```
                    ┌─YES─┐
►►──SEQ=──┬───┬─────┴─NO──┴──────────────────────────────────────────►◄
          └─*─┘
```

Default
SEQ=YES

YES
Checks that the source statements are in ascending alphanumeric order by line number.

NO
Does not perform sequence checking.

Notes:

1. If both SEQ and NUM are in effect at compile time, the sequence is checked according to numeric, rather than alphanumeric, collating sequence.

SIZE

Syntax

```
                     ┌─MAX──────┐
►►──SIZE=──┬───┬─────┼─integer──┼─────────────────────────────────────►◄
           └─*─┘     └─integerK─┘
```

Default
SIZE=MAX

integer
Specifies the amount of virtual storage available, in bytes. The minimum acceptable value is 778240.

integerK
Specifies the amount of virtual storage available in 1024-byte (K) increments.

The minimum acceptable value is 760-KB.

MAX
Requests all available space in the user region for use during the compilation. For extended architecture, the compiler obtains the largest contiguous block of free storage above the 16-MB line.

Notes:

1. Using SIZE=MAX simplifies compiler invocation by eliminating the need to determine a specific value for the SIZE option.

2. Do not use SIZE=MAX if, when you invoke the compiler, you require it to leave a specific amount of unused storage available in the user region.
3. SIZE=MAX in Extended Architecture allows the compiler to obtain all available above-the-line storage in the user region and below-the-line storage for work file buffers and compiler modules that must be below the 16-MB line. This allocation occurs unless tuning is done for the extended architecture environment. Therefore, you might not want to fix this option at SIZE=MAX at installation.

SOURCE

Syntax

```
                   ┌─YES─┐
►►──SOURCE=──┬───┬─┴─NO──┴──────────────────────────────────────────►◄
             └─*─┘
```

Default
SOURCE=YES

YES
Indicates that you want a listing of the source statements in the compiler-generated output. This listing also includes any statements embedded by COPY.

NO
Source statements do not appear in the output.

Note: The SOURCE compiler option must be in effect at compile time if you want embedded messages in the source listing.

SPACE

Syntax

```
                  ┌─1─┐
►►──SPACE=──┬───┬─┼─2─┼─────────────────────────────────────────────►◄
            └─*─┘ └─3─┘
```

Default
SPACE=1

1 Provides single spacing for the source statement listing.

2 Provides double spacing for the source statement listing.

3 Provides triple spacing for the source statement listing.

SQL

Syntax

```
                ┌─NO──┐
►►──SQL=──┬───┬─┴─YES─┴─────────────────────────────────────────────►◄
          └─*─┘
```

Default
SQL=NO

NO
Specify to have any SQL statements found in the source program diagnosed and discarded.

Use SQL=NO if your COBOL source programs do not contain SQL statements, or if the seperate SQL preprocessor will be used to process SQL statements before invocation of the COBOL compiler.

YES
Use to enable the DB2 coprocessor capability and to specify DB2 suboptions. You must specify the SQL option if your COBOL source program contains SQL statements and it has not been processed by the DB2 precompiler.

Notes:

1. You can specify the SQL option in any of the compiler option sources: compiler invocation, PROCESS/CBL statements, or installation defaults.
2. Use either quotes or apostrophes to delimit the string of DB2 suboptions.
3. DB2 suboptions cannot be specified as part of customizing the SQL option. (DB2 suboptions are supported only when the SQL compiler option is specified as an invocation option or on a CBLor PROCESS card.) However, default DB2 options can be specified when you customize the DB2 product installation defaults.
4. The SQL=YES option conflicts with the LIB=NO option.

SSRANGE

Syntax

```
                         ┌─NO──┐
►►──SSRANGE=──┬───┬──────┴─YES─┴──────────────────────────────────►◄
              └─*─┘
```

Default
SSRANGE=NO

YES
Generates code that checks subscripts, reference modifications, variable-length group ranges, and indexes in the source program at run time to ensure that they do not refer to storage outside the area assigned. It also verifies that a table with ALL subscripting, specified as a function argument, contains at least one occurrence in the table.

The generated code also checks that a variable-length item does not exceed its defined maximum length as a result of incorrect setting of the OCCURS DEPENDING ON object.

Performance consideration: If SSRANGE=YES at compile time, object code size is increased and there will be an increase in run-time overhead to accomplish the range checking.

NO
No code is generated to perform subscript or index checking at run time.

Notes:

1. If the SSRANGE option is in effect at compile time, the range-checking code is generated.
2. Range-checking can be inhibited at run time by specifying the Language Environment run-time option CHECK(OFF). However, the range-checking code still requires overhead and is dormant within the object code.
3. The range-checking code can be used optionally to aid in resolving any unexpected errors without recompilation.

TERM

Syntax

```
                   ┌─NO──┐
►►──TERM=──┬───┬───┴─YES─┴──────────────────────────────────────────────►◄
           └─*─┘
```

Default
TERM=NO

YES
Specifies that the progress and diagnostic messages are sent to the SYSTERM file, which defaults to the user's terminal unless specified otherwise.

NO
Specifies that no messages are sent to the SYSTERM file.

Note: If TERM is specified in the source program, a SYSTERM DD statement must also be specified for each application program.

TEST

Syntax

```
                   ┌─NO──────────────────────────────────────┐
►►──TEST=──┬───┬───┴─(──hook──,──symbol──,──inclusion──)─────┴──────────►◄
           └─*─┘
```

Default
TEST=NO

Other than NO
Produces object code that can be run using Debug Tool.

You must specify values for the hook, symbol and inclusion suboptions:

hook values:

ALL Activates the generation of all compiled-hooks. Hooks are generated at all statements, all path points, and all program entry and exit points. In addition, if either the DATEPROC=FLAG option or DATEPROC=NOFLAG option is in effect, hooks are generated for all date-processing statements.

NONE
Suppresses the generation of all compiled hooks. TEST(NONE) is compatible with the OPT compiler option.

STMT Hooks are generated at every statement and label, as well as at all program entry and exit points. In addition, if either the DATEPROC=FLAG option or DATEPROC=NOFLAG option is in effect, hooks will be generated for all date-processing statements.

PATH Hooks are generated at all path points, including program entry and exit points.

BLOCK
Hooks are generated at all program entry and exit points.

symbol values:

SYM Activates generation of symbolic dictionary information tables.

NOSYM
Deactivates generation of symbolic dictionary information tables.

inclusion values:

SEPARATE
Generate the debugging information tables in a data set separate from your object program. You can specify SEPARATE only if you also specify SYM as the value for the symbol sub-option.

NOSEPARATE
Include the debugging information tables in your object program.

Performance consideration: Because TEST=(a hook-location suboption other than NONE) generates additional code, it can cause significant performance degradation at run time when used in a production environment.

NO
Produces object code that cannot be symbolically debugged using Debug Tool.

Notes:

1. If you specify TEST= (other than NONE for the hook-location suboption), the following options are put into effect at compilation time:
   OBJ=YES
   OPT=NO
2. Only hook location NONE is compatible with the OPT compiler option. See “Conflicting compiler options” on page 13 for more conflict resolution information.
3. Programmers might notice an increase in run time for programs compiled with any hook-location suboption other than NONE.
4. A date-processing statement is any statement that references a date field, or any EVALUATE or SEARCH statement WHEN phrase that references a date field.
5. For production programs, compile your programs with TEST(NONE,SYM,SEPARATE) if you do not want to increase the size of your production modules, but still get minimum debugging capability.

THREAD

Syntax

```
                      ┌─NO──┐
►►──THREAD=──┬───┬────┴─YES─┴─────────────────────────────────────►◄
             └─*─┘
```

Default
THREAD=NO

NO
Use NO if you want to indicate that a COBOL program is not enabled for execution in a Language Environment enclave with multiple POSIX threads or PL/I tasks.

YES
Use THREAD if you want to indicate that a COBOL program is enabled for execution in a Language Environment enclave with multiple POSIX threads or PL/I tasks.

Notes:

1. When the THREAD compiler option is specified, the program is enabled for use in a threaded application. However, it can still be used in nonthreaded applications. For example, you can run a program that was compiled with the THREAD option in the CICS environment, if the application does not contain multiple POSIX threads or PL/I tasks at run time.
2. When the THREAD compiler option is specified, run-time performance might be degraded because of the serialization logic that is automatically generated.
3. When the THREAD compiler option is specified, the RENT compiler option must also be specified. If THREAD and NORENT are specified at the same level of precedence, the RENT option is forced on.
4. For COBOL programs to run in a threaded application, all COBOL programs in the run unit must be compiled with the THREAD compiler option specified.
5. When the THREAD compiler option is specified, the following language elements are not supported. If any of the following language elements are specified, they are diagnosed as errors:
   - Nested programs
   - INITIAL phrase in the PROGRAM-ID clause
   - ALTER statement
   - DEBUG-ITEM special register
   - GO TO statement without a procedure name
   - RERUN
   - STOP literal statement
   - Segmentation module
   - USE FOR DEBUGGING statement
   - WITH DEBUGGING MODE clause
   - SORT or MERGE statements
6. When you compile programs with the THREAD compiler option, the following special registers are allocated upon each invocation:
   - ADDRESS-OF
   - RETURN-CODE

- SORT-CONTROL
- SORT-CORE-SIZE
- SORT-FILE-SIZE
- SORT-MESSAGE
- SORT-MODE-SIZE
- SORT-RETURN
- TALLY
- XML-CODE
- XML-EVENT

TRUNC

Syntax

```
                        ┌─STD─┐
►►──TRUNC=──┬───┬──┼─OPT─┼──────────────────────────────────────►◄
            └─*─┘      └─BIN─┘
```

Default
TRUNC=STD

STD
Conforms to the COBOL 85 Standard.

Controls the way arithmetic fields are truncated during MOVE and arithmetic operations. The TRUNC option applies only to binary (COMP) receiving fields in MOVE statements and in arithmetic expressions. When TRUNC=STD is in effect, the final intermediate result of an arithmetic expression, or of the sending field in the MOVE statement, truncates to the number of digits in the PICTURE clause of the binary receiving field.

OPT
The compiler assumes that the data conforms to PICTURE and USAGE specifications. The compiler manipulates the result based on the size of the field in storage (halfword or fullword).

TRUNC=OPT is recommended, but it should be specified only when data being moved into binary areas does not have a value with larger precision than that defined by the binary item PICTURE clause. Otherwise, truncation of high-order digits might occur. The truncation results are dependent on the particular code sequence generated and might not be the same in OS/VS COBOL and Enterprise COBOL.

BIN
Should not be used as an installation default. Specifies that:

1. Output binary fields are truncated only at the S/390 halfword, fullword, and doubleword boundaries, rather than at COBOL base 10 picture limits.
2. Input binary fields are treated as S/390 halfword, fullword, and doubleword, and no assumption is made that the values are limited to those implied by the base 10 PICTURE clause.
3. DISPLAY converts and outputs the full content of binary fields with no truncation to the PICTURE description.

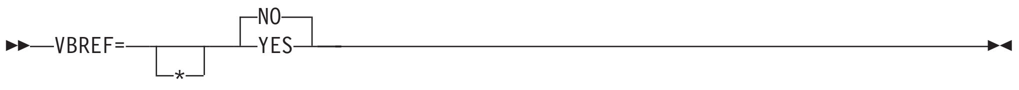
Performance consideration: Using TRUNC=OPT does not generate extra code and generally improves performance. However, both TRUNC=BIN and TRUNC=STD generate extra code whenever a BINARY data item is changed. TRUNC=BIN is usually the slower of these options.

Notes:

1. Setting this option affects program run-time logic; that is, the same COBOL source program can give different results, depending on the option setting. Verify whether your COBOL source programs assume a particular setting for correct running.
2. TRUNC=BIN is the recommended option when interfacing with other products that have S/390-format binary data (such as CICS, DB2, FORTRAN, and PL/I). This is especially true if there is a possibility of having more than 9 digits in a fullword or more than 4 digits in a halfword.

VBREF

Syntax



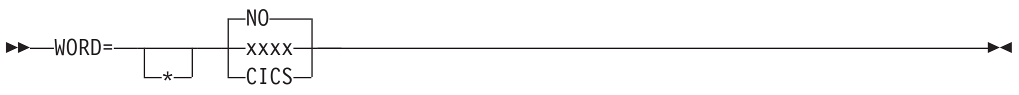
Default
VBREF=NO

YES
Produces a cross-reference of all verb types in a source program to the line numbers where they are found. VBREF=YES also produces a summary of how many times each verb was used in the program.

NO
Does not produce a cross-reference or verb-summary listing.

WORD

Syntax



Default
WORD=*NO

NO
Indicates that no alternative reserved word table is to be used as the default.

xxxx
Specifies an alternative default reserved word table to be used during compilation. **xxxx** represents the ending characters (can be 1 to 4 characters in length) of the name of the reserved word table used. The first 4 characters are IGYC. The last 4 characters cannot be any one of the character strings listed below, nor can any of them contain the dollar sign character ($).

ASM1	LIBO	LVL8	RDSC
ASM2	LIBR	OPTM	RWT
DIAG	LSTR	OSCN	SAW
DMAP	LVL0	PGEN	SCAN
DOPT	LVL1	RCTL	SIMD
FGEN	LVL2	RDPR	XREF
INIT	LVL3		

CICS
A CICS-specific word table, IGYCCICS, is provided as an alternate reserved word table. For a description, see “CICS reserved word table (IGYCCICS)” on page 10.

Notes:

1. The default for the WORD option is specified with an asterisk. When the option is installed with the default (WORD=*NO), the application programmer cannot override the option at compile time to specify an alternative reserved word table.
2. Specification of WORD affects the interpretation of input reserved words. System names (such as UPSI and SYSPUNCH) and the 42 intrinsic function names should not be used as aliases for reserved words. If a function name is specified as an alias through the reserved word table ABBR control-statement, that function name will be recognized and diagnosed by the compiler as a reserved word and the intrinsic function will not be performed.
3. Changing the default value of the WORD=XXXX option conflicts with all values for FLAGSTG other than no.

XREFOPT

Syntax

```
                         ┌─NO────┐
►►──XREFOPT=──┬───┬──────┼─SHORT─┼──────────────────────────────────►◄
              └─*─┘      └─FULL──┘
```

Default
XREFOPT=FULL

SHORT
Produces only the explicitly referenced variables in the cross-reference listing.

FULL
Produces both a sorted and embedded cross-reference listing. If SOURCE=YES is also specified, the listing line numbers cross-reference recurrences of particular data-names.

NO
Suppresses the cross-reference listing.

Notes:

1. The XREFOPT option sets the default value for the compiler option XREF.
2. The XREFOPT=NO option conflicts with values for DBCSXREF other than NO.

YRWINDOW

Syntax

```
                         ┌─1900────┐
►►──YRWINDOW=─────────────┴─integer─┴──────────────────────────────────────►◄
              └─*─┘
```

Default
YRWINDOW=1900

integer
Specifies the first year of the 100-year window used by COBOL windowed date fields, and may be one of the following:

- An unsigned decimal integer from 1900 through 1999.
- A negative decimal integer from -1 through -99, representing an offset from the current year at run time. The current year is determined for each compilation unit when it is first initialized, or reinitialized after execution of a CANCEL statement that refers to the compilation unit.

Notes:

1. YRWINDOW has no effect unless DATEPROC=FLAG or DATEPROC=NOFLAG is specified.
2. At run time, two conditions must be true:
   a. The 100-year window must have its beginning year in the 1900s.
   b. The current year must lie within the 100-year window for the compilation unit.
3. All windowed dates have a year relative to the base year. For example, if the base year were specified as 1965, all windowed year values would be interpreted as years within the 100-year window of 1965 to 2064, inclusive. So, a windowed year value of 67 would represent the year 1967, whereas a windowed year value of 05 would represent the year 2005.

   If the base year were specified as -30, then the window would depend on when the application were run. Running it during 2000 would give a 100-year window of 1970 through 2069. So, a windowed year value of 70 would represent the year 1970, whereas a windowed year value of 69 would represent the year 2069.
4. The YRWINDOW installation option is equivalent to the YEARWINDOW compile-time option.

ZWB

Syntax

```
                  ┌─YES─┐
►►──ZWB=───────────┴─NO──┴──────────────────────────────────────────────────►◄
         └─*─┘
```

Default
ZWB=YES

YES
Removes the sign from a signed external decimal (DISPLAY) field when comparing this field to an alphanumeric field at run time.

NO
Does not remove the sign from a signed external decimal (DISPLAY) field when comparing this field to an alphanumeric field at run time.

Notes:

1. Setting this option affects program run-time logic; that is, the same COBOL source program can give different results, depending on the option setting. Verify whether your Enterprise COBOL source programs assume a particular setting to run correctly.
2. Application programmers use ZWB=NO to test input numeric fields for SPACES.

Enterprise COBOL compiler options

Chapter 3. Customizing Enterprise COBOL

You can make modifications to Enterprise COBOL only after installation of the product is complete. One of the modifications is made using an SMP/E USERMOD. If you do not ACCEPT Enterprise COBOL into the distribution libraries before applying the USERMOD, you will not be able to use the SMP/E RESTORE statement to remove your USERMOD. Do not accept your USERMOD into the distribution libraries. You might want to remove your USERMOD if you find that it does not suit the needs of the programmers at your site.

You will have to remove your USERMOD before applying service to the modules that it changes. In this case, you will probably want to reapply your USERMOD after successful installation of the service.

Important

Make sure that Enterprise COBOL serves the needs of the application programmers at your site. Confer with them while you plan the customization of Enterprise COBOL. Doing so will ensure that the modifications you make at install time best support the application programs being developed at your site.

Note: All information for installing Enterprise COBOL is included in the Program Directory provided with the product.

Summary of user modifications

Installation of Enterprise COBOL places a number of sample modification jobs in the target data set IGY.V3R2M0.SIGYSAMP. Table 7 shows the names of the sample modification jobs, which are described in detail in the following sections.

The sample modification jobs that IBM provides are not customized for your particular system. You must customize them.

Copy members from IGY.V3R2M0.SIGYSAMP into one of your personal data sets before you modify and submit them so that you have an unmodified backup copy if you make changes that you want to abandon.

Descriptions of possible modifications appear in the comments in the JCL. You can use TSO to modify and submit the job. Save the modified JCL for future IGYWMLPA reference.

Table 7. Summary of user modification jobs for Enterprise COBOL

Description	Customization job	Page
Change compiler options default module	IGYWDOPT	54
Create an options module to override compiler options specified as fixed	IGYWUOPT	55
Create additional reserved word table	IGYWRWD	56
Place Enterprise COBOL modules in shared storage	IGYWMLPA	61

Changing the defaults for compiler options

To change the defaults for compiler options or create a compiler options module to override fixed options, copy the source of options module IGYCDOPT from IGY.V3R2M0.SIGYSAMP into the appropriate job in place of the two-line comment. Then change the parameters on the IGYCOPT macro call to match the compiler options that you have selected for your installation. Observe the following requirements in coding your changed IGYCOPT macro call:

- Place continuation character (X in the source) must be present in column 72 on each line of the IGYCOPT invocation except the last line. The continuation line must start in column 16. You can break the coding after any comma.
- Do not put a comma in front of the first option in your macro.
- Specify options and suboptions in uppercase. Only suboptions that are strings can be specified in mixed-case or lowercase. For example both LVLINFO=(Fix1) and LVLINFO=(FIX1) are acceptable.
- If one of the string suboptions contains a special character, (for example, an embedded blank or unmatched right or left parenthesis), the string must be enclosed in apostrophes (′), not in quotes (″). A null string can be specified with either contiguous apostrophes or quotes.

  To obtain an apostrophe (′) or a single ampersand (&) within a string, two contiguous instances of the character must be specified. The pair is counted as only one character in determining whether the maximum allowable string length has been exceeded and in setting the effective length of the string.
- Avoid unmatched apostrophes in any string that uses apostrophes. The error cannot be captured within IGYCOPT itself. Instead, the assembler produces a message such as:

  ```
  IEV03  *** ERROR ***  NO ENDING APOSTROPHE
  ```

  This message bears no spatial relationship to the offending suboption. Furthermore, none of the options are properly parsed if this error is committed.
- Code only the options whose default value you want to change. The IGYCOPT macro supplies the IBM-supplied defaults for any option that you do not code. See “IGYCDOPT worksheet for compiler options” on page 1-4 for a worksheet to help you plan your default compiler options. See “Enterprise COBOL compiler options” on page 1-24 for descriptions of the options.
- Place an END statement after the macro instruction.

Changing compiler options default module

Use the sample job IGYWDOPT to change the defaults for the Enterprise COBOL compiler options. Use the information in Chapter 2, “Enterprise COBOL compiler options” on page 13 to select your default values.

If you coded OUT as the value for any compiler phase options, be sure to place these phases in shared storage before compiling a program by using your new compiler options default module. See “Compiler phases and their defaults” on page 6 and “Placing Enterprise COBOL modules in shared storage” on page 61 for more information.

To modify the JCL for IGYWDOPT, do these steps:

1. Add a job card appropriate for your site.
2. Add a JES ROUTE card if required for your site.
3. Replace the two comment lines in IGYWDOPT with a copy of the source for IGYCDOPT found in IGY.V3R2M0.SIGYSAMP.
4. Code parameters on the IGYCOPT macro statement in IGYCDOPT to reflect the values you have chosen for your installation-wide default compiler options.
5. Change #GLOBALCSI to the global CSI name.
6. Change #TZONE in the SET BDY statement to the target zone name.

After you modify the IGYWDOPT job, submit it. You will get a condition code of 0 if the job runs correctly. Also check the IGYnnnn informational messages in your listing to verify the defaults that will be in effect for your installation.

Creating an options module to override options specified as fixed

If you have specified some options as fixed in your compiler default options module, you might occasionally find an application that needs to override a fixed option. You can provide other options by creating a temporary copy of the options module in a separate data set that can be accessed as a STEPLIB or JOBLIB (ahead of the IGY.V3R2M0.SIGYCOMP data set) when the application is compiled. Sample job IGYWUOPT creates a default options module that is link-edited into a user-specified data set. The assembly and link-editing take place outside SMP/E control, so the standard default options module is not disturbed.

To modify the JCL for IGYWUOPT, do these steps:

1. Add a job card appropriate for your site.
2. Add a JES ROUTE card if required for your site.
3. Replace the two comment lines in IGYWUOPT with a copy of the source for IGYCDOPT found in IGY.V3R2M0.SIGYSAMP.
4. Change the parameters on the IGYCOPT macro statement in IGYWUOPT to reflect the values that you have chosen for this fixed option override compiler options module.
5. If you chose to use a different prefix than the IBM-supplied one for the Enterprise COBOL target data sets, check the SYSLIB DD statement (marked with '<<<<<') to ensure that the data set names are correct.
6. Change DSNAME=YOURLIB in the SYSLMOD DD statement to the name of the partitioned data set that you want your IGYCDOPT module linked into. Note that an IGYCDOPT module currently in the chosen data set will be replaced by the new version.

After you modify the IGYWUOPT job, submit it. Both steps should return a condition code of 0 if the job runs successfully. Also check the IGYnnnn informational messages in your listing to verify the defaults that are in effect when this module is used in place of the standard default options module.

Creating or modifying additional reserved word tables

The reserved words used by the Enterprise COBOL-compiler are maintained in a table (IGYCRWT) provided with the product. A CICS-specific reserved word table (IGYCCICS) is provided as an alternate reserved word table. See "CICS reserved word table (IGYCCICS)" on page 10. You can change the reserved words by using the reserved word table utility (IGY8RWTU), thereby creating additional reserved word tables. You can also modify tables that you previously created.

The reserved word table utility accepts control statements that you can use to create or modify a reserved word table. The new table then contains the reserved words from the IBM-supplied table with all the changes that you have applied.

You can make the following types of changes to reserved word tables:
- Add an alternative form of an existing reserved word.
- Add words to be flagged with an informational message whenever they are used in a program.
- Add words to be flagged with an error message whenever they are used in a program.
- Indicate that words currently flagged with an informational or error message should no longer be flagged.

Each reserved word table that you create must have a unique 1- to 4-character identifier. For a list of 1- to 4-character strings that cannot be used, see the compiler options ″WORD=xxxx″.

At compile time, the value of the compiler option **WORD(xxxx)** identifies the reserved word table to be used. **xxxx** is the unique 1- to 4-character identification that you specified in the member name IGYCxxxx. You can create multiple reserved word tables, but only one can be specified at compile time.

Note: The total number of entries in a reserved word table should not exceed 1536 or 1.5-KB.

Creating or modifying a reserved word table

To create or modify a reserved word table, you must edit a copy of the appropriate source file:
- Member IGY8RWRD in IGY.V3R2M0.SIGYSAMP (the IBM-supplied default reserved word table)
- Member IGY8CICS in IGY.V3R2M0.SIGYSAMP (the IBM-supplied CICS reserved word table)
- A user file (user-supplied reserved word table)

You must also modify and invoke the appropriate non-SMP/E JCL.

Your file should have four parts: Parts I, II, III, and IV. Modify the file and non-SMP/E JCL as follows:
1. Make a private copy of the file.
2. Do not edit Part I (all lines up to and including the line with the keyword MOD).
3. Edit Part II by placing asterisks in column 1 of the lines that contain CODASYL reserved words for which you do not want informational messages issued.

4. Edit Part III by placing asterisks in column 1 of the lines that contain obsolete reserved words for which you do not want severe messages issued.
5. Edit Part IV by coding additional reserved word control statements that create the modifications that you want, as described under "Coding control statements".
6. Modify and run the JCL, as discussed under "Modifying and running JCL to create a new reserved word table" on page 60. You also must create a unique 1- to 4-character identification for the new reserved word table and supply it in the JCL.

Coding control statements

To create a reserved word table, you must understand the syntax rules for the control statements and for the operands within control statements.

Figure 2 illustrates the format for coding reserved word processor control statements.

```
ABBR   reserved-word: user-word [comments]
      [reserved-word: user-word [comments]]

⋮
INFO   COBOL-word [(0 | 1)] [comments]
      [COBOL-word [(0 | 1)] [comments]]

⋮
RSTR   COBOL-word  [(0 | 1)] [comments]
      [COBOL-word [(0 | 1)] [comments]]

⋮
```

Figure 2. Syntax format for reserved word processor control statements

As shown in Figure 2, keywords you can use are:

ABBR Specifies an alternative form of an existing reserved word

INFO Specifies words that are to be flagged with an informational message whenever they are used in a program

RSTR Specifies words that are to be flagged with an error message whenever they are used in a program

Note: All words that you identify with the control statement keywords INFO and RSTR are flagged with a message in the source listing of the Enterprise COBOL program that uses them. Words that are abbreviated are not flagged in the source listing unless you have also specified them on the INFO or RSTR control statements.

Rules for coding control statements

When you code your control statements, follow these rules:

- Begin the control statement in column 1.
- Place one or more spaces between the keyword and the first operand.
- When specifying a second operand, include a colon (:) and one or more spaces after the first operand.

- Continue a control statement by putting blanks in columns 1 through 5, followed by the operand or operands, to make additional specifications.
- Specify comments by putting an asterisk (*) in column 1 of the control statements. You can also place comments on the same line as the control statement. In that case, however, there must be at least one space following the operand or operands before a comment begins.
- To specify more than one change within a single control statement, put each additional specification on a separate line.
- Do not add any blank lines.

Coding operands in control statements

The following list shows the types of operands that you will be coding in the control statements:

reserved-word
An existing reserved word.

user-word
A user-defined COBOL word that is not a reserved word.

comments
Any comments that you want to put on the same line with the control statement, or on a separate line that has an asterisk in column 1.

COBOL-word
A word of up to 30 characters that can be a system name, a reserved word, or a user-defined word.

Rules for coding control statement operands

When you code the control statement operands, follow these rules:

- A user-word can be used in only one ABBR statement in any particular reserved word table.
- A reserved-word specified in an ABBR statement can also be specified in either a RSTR or an INFO statement.
- A particular reserved-word can be specified only once in an ABBR statement.
- A particular COBOL-word can be specified only once in either a RSTR or an INFO statement.

The remaining sections provide examples for coding each type of control statement.

ABBR statement

ABBR reserved-word: user-word [comments]
Defines an alternative symbol for the reserved word specified. The symbol can be used when you code a program.

Notes:

1. The user-word becomes a reserved word and can be used in place of the reserved-word specified in this statement.
2. The reserved-word remains a reserved word with its original definition.
3. The source listing shows the original source—the symbol as you coded it.
4. The reserved word is used in compiler output—other listings, some messages, and so forth.

In the following example, REDEF and SEP become abbreviations that can be used in source programs. The appropriate reaction to the use of REDEFINES and SEPARATE takes place when the source program is compiled.

```
ABBR   REDIFINES: REDEF
SEPARATE:   SEP
```

INFO statement

INFO COBOL-word[(0 | 1)] [comments]

This statement specifies the COBOL words that are to be flagged by the compiler.

This statement can also be used to control the use of nested programs. By selecting either 1 or 0, you can indicate whether a specific COBOL-word can be used only once, or not at all.

0 Indicates that whenever the specified COBOL-word is used, the 0086 informational message is issued.

1 Indicates that the specified COBOL-word can be used once. If it is used more than once, informational message 0195 is issued.

The messages are handled as information (I) messages. The compilation condition is not changed.

Because of the following example, when the IBM extension reserved word ENTRY is used in a program, it will be flagged with message 0086.

```
INFO  ENTRY
```

RSTR statement

RSTR COBOL-word[(0 | 1)] [comments]

This statement specifies COBOL words that cannot be used in a program.

This statement can also be used to control the use of nested programs. By selecting either 1 or 0, you can indicate whether a specific COBOL-word can be used only once, or not at all.

0 Indicates that whenever the specified COBOL-word is used, message 0084 is issued.

1 Indicates that the specified COBOL-word can be used once. If it is used more than once, severe message 0194 is issued.

Note: The following reserved words can be restricted with the 1 option only:

IDENTIFICATION
FD
ENVIRONMENT
DATA
WORKING-STORAGE
PROCEDURE
DIVISION
SECTION
PROGRAM-ID

The following example restricts the use of Boolean, XD, and PARENT. Use of these will cause errors.

```
RSTR  BOOLEAN
      XD
      PARENT
```

The following example restricts the use of GO TO and ALTER. Use of these will cause errors.

```
RSTR  GO
      ALTER
```

In the following example, the reserved word table generated allows usage of all COBOL 85 Standard language except nested programs.

```
RSTR IDENTIFICATION(1)  only allow 1 program per compilation unit
RSTR ID(1)              same for the short form
RSTR PROGRAM-ID(1)      only allow 1 program per compilation unit
RSTR GLOBAL             do not allow this phrase at all
```

Modifying and running JCL to create a new reserved word table

The JCL that you use to create a new reserved word table contains STEP1, STEP2, and STEP3, which do the following, respectively:

Run the reserved word table utility with your modified table.
Assemble your modified reserved word table.
Produce a run-time load module from the object module.

After you run the job, a new reserved word table is created, the library that you specified contains the new table, and the table has IGYC plus the 1- to 4-character identification that you specified.

Modifying and running non-SMP/E JCL

Use sample job IGYWRWD in IGY.V3R2M0.SIGYSAMP to create your new reserved word table. The sample job is distributed with member IGY8---- in IGY.V3R2M0.SIGYSAMP as the input to the reserve word utility, and creates load module IGYC---- in IGY.V3R2M0.SIGYCOMP. Before you run the job, do the following:

To Modify the JCL for IGYWRWD, do these steps:

1. Modify the job statement for your site.
2. Add JES ROUTE records if desired.
3. Change the data set name on the STEPLIB DD statement in STEP1 to match the compiler target data set name you used during installation.
4. Do *one* of the following steps to point to your modified reserved word table:
   - Change the data set name in //RSWDREAD DD DSN=... to the data set name and member name of your modified reserved word table, or
   - Replace the RSWDREAD DD with //RSWDREAD DD * and insert your modified reserved table immediately following that line.

   Note: For specific instructions, see the comments in job IGYWRWD.
5. Change the name of the data set on the SYSLMOD DD statement in STEP3 to match the name of the data set to which you are adding your modified reserved word table. (The data set name on the SYSLMOD DD statement should be the name of the compiler target data set.) Also, you must specify the name of your modified reserved word table in the parentheses that follow the data set name on the SYSLMOD DD statement.

If, after you run IGYWRWD, you receive a nonzero return code from the table utility, use the error messages in the output data set specified on the RSWDPRNT DD statement to correct any mistakes and rerun the job.

Placing Enterprise COBOL modules in shared storage

All of the modules in IGY.V3R2M0.SIGYCOMP that are reentrant can be included in shared storage by:

- Authorizing the data set IGY.V3R2M0.SIGYCOMP
- Including IGY.V3R2M0.SIGYCOMP in the LNKLSTnn concatenation (optional)
- Creating an IEALPAnn member in SYS1.PARMLIB that lists the modules to be made resident in the MLPA when the system is IPLed

IGYWMLPA is installed in IGY.V3R2M0.SIGYSAMP for you to use as an example in creating your IEALPAnn member.

Under OS/390 or z/OS, you do not need to place IGY.V3R2M0 in the LNKLSTnn concatenation to be able to load modules into the LPA. If you choose not to add it to the LNKLSTnn concatenation, you must make the modules that are not included in the LPA available to steps that compile Enterprise COBOL applications by *one* of these means:

- Copying the non-LPA modules to a data set that is in the LNKLSTnn concatenation
- Copying the non-LPA modules to a data set that can be used as a STEPLIB or JOBLIB

Using the entire IGY.V3R2M0.SIGYCOMP data set as a STEPLIB or JOBLIB defeats the purpose of placing the modules in the LPA because modules are loaded from a STEPLIB or JOBLIB before the LPA is searched.

Modules that you copy into another data set are not serviced automatically by SMP/E in that data set. You must rerun your copy job after you apply service to Enterprise COBOL to make the updated modules available in the LNKLSTnn data set or in the STEPLIB.

Refer to the following publications for more information about including modules in the LPA:

- *OS/390 MVS Initialization and Tuning Guide*, SC28-1751
- *z/OS MVS Initialization and Tuning Guide, SA22-7591*
- *OS/390 MVS Initialization and Tuning Reference*, SC28-1752
- *z/OS MVS Initialization and Tuning Reference, SA22-7592*

If you are placing compiler phases in shared storage, code the corresponding phase options with the value OUT when you run the sample job IGYWDOPT to change the compiler options defaults. See "Changing the defaults for compiler options" on page 54 for more information.

Tailoring the cataloged procedures to your site

You might want to tailor the cataloged procedures IGYWC, IGYWCL, IGYWCLG, IGYWCG, IGYWCPL, IGYWCPLG, IGYWCPG, and IGYWPL for use at your site. Changes to consider are:

- Modifying the data set name prefixes if you chose to use a different prefix than the IBM-supplied ones for Enterprise COBOL or Language Environment target data sets.
- Removing the STEPLIB DD statements if you have placed IGY.V3R2M0.SIGCOMP and CEE.SCEERUN in the LINKLIST concatenation.
- Changing the default region size for the GO steps if most of the programs at your site require a larger region for successful execution.
- Changing the UNIT=parameter.

Chapter 4. Customizing Unicode support for COBOL

In Enterprise COBOL, a national data type, national literals, intrinsic functions, and compiler options provide basic run-time support for Unicode. Also, object-oriented syntax for Java interoperability uses Unicode capabilities implicitly. COBOL source programs continue to be encoded in an EBCDIC (SBCS or DBCS) code page.

Before you can compile or run the following types of COBOL programs, you must install and configure Unicode conversion services on both development systems (where programs are compiled) and production systems (where programs are run).

- COBOL programs that use object-oriented syntax to interoperate with Java
- COBOL programs that contain national data types, national literals, or DISPLAY-OF or NATIONAL-OF intrinsic functions

If the required conversion support is not available, a severity-3 Language Environment condition is raised at run time, or a compiler diagnostic is issued.

Installing, setting up, and activating Unicode conversion services

Follow the instructions in the following documents for the operating system that you use:

- For z/OS, Version 1 Release 2 or later, z/OS Support for Unicode is included as part of the operating system. See *z/OS Support for Unicode: Using Conversion Services* (SA22-7649).
- For z/OS Version 1 Release 1, or OS/390, Version 2 Release 10, you need to install OS/390 Support for Unicode by downloading the Support for Unicode (FMID HUNI2A0) software from the Web as follows:
  1. Open a Web browser and go to the Enterprise COBOL support page at http://www.ibm.com/software/ad/cobol/zos/support/.
  2. Click **OS/390 Support for Unicode** under Support downloads.

  For instructions on installing OS/390 2 R8/R9/R10 Support for Unicode, refer to the program directory (GI10-9760) at:
  - http://publib.boulder.ibm.com/pubs/pdfs/os390/cunpde00.pdf (for PDF format)
  - http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/cunpde00/ (for BookManager format)

Creating a conversion image for COBOL

After the system has been set up with Unicode conversion services, you must create and activate a conversion image that supports the use of Unicode conversion services on the system. This section describes how to create a conversion image that supports COBOL requirements.

To create a conversion image, invoke the conversion image generator with control statements, each of which results in the inclusion of the following items:

- The conversion table for the specified source CCSID and target CCSID
- The conversion technique that is to be used for the conversion

The following CCSID pairs are required for COBOL programs, depending on the language features used:

Program description	Source CCSID	Target CCSID
The program uses Unicode data.	CCSID that is in effect through the CODEPAGE compiler option	CCSID 1200 (UTF-16)
	CCSID 1200	CCSID that is in effect in the CODEPAGE compiler option
The program contains the NATIONAL-OF intrinsic function with an explicit CCSID specification.	CCSID that is specified in the intrinsic function	CCSID 1200
The program contains the DISPLAY-OF intrinsic function with an explicit CCSID specification.	CCSID 1200	CCSID that is specified in the intrinsic function
The program contains object-oriented syntax for Java interoperability.	CCSID that is in effect through the CODEPAGE compiler option	CCSID 1208 (UTF-8)
	CCSID 1200	CCSID 1208

The conversion technique is called *technique search order*. COBOL uses the technique search order RECLM, which stands for roundtrip, enforced subset, customized, Language Environment-behavior, and modified language. RECLM is the default technique search order, so you can omit the technique search order on the control statements . The selection of the technique search order that COBOL uses is fixed, and application programs cannot change it.

Except for the the CCSID pairs required for object-oriented syntax, the CCSID pairs are always the combination of CCSID 1200 and another CCSID.

When a program contains object-oriented syntax for Java interoperability, the following CCSID pairs are required:
- CCSID that is specified through the CODEPAGE compiler option to CCSID 1208
- CCSID 1200 to CCSID 1208

The default that IBM ships for the CODEPAGE compiler option is 1140 (EBCDIC Latin-1 with the euro symbol.

Example: programs that use object-oriented syntax for Java interoperability

Consider an installation that has applications with the following characteristics:
- They use only the default value of the CODEPAGE compiler option (which is CCSID 1140).
- They do not specify CCSID values explicitly on the DISPLAY-OF or NATIONAL-OF intrinsic functions.
- They use object-oriented syntax for Java interoperability.

In this case, configure the Unicode conversion services with (at least) the following set of conversions:

- CCSID 1140 to 1200
- CCSID 1200 to 1140
- CCSID 1140 to 1208
- CCSID 1200 to 1208

Example: program that uses Unicode data

The following example uses national data items along with the DISPLAY-OF and NATIONAL-OF intrinsic functions (the text in **bold** is not part of the source program):

```
CBL CODEPAGE(1140) NSYMBOL(NATIONAL)
Identification Division.
Program-ID. Sample1.
. . .
Data Division.
Working-Storage Section.
01 Latin1-data PIC X(10).
01 ASCII-Latin1-data PIC X(10).
01 Japanese-MBCS-data PIC X(20).
01 National-data PIC N(10) Usage National.
. . .
Procedure Division.
    . . .
(1) Move 'ABCDE12345' to Latin1-data
(2) Move Latin1-data to National-data
(3) Move Function DISPLAY-OF(National-data,1252)
      To ASCII-Latin1-data
(4) Move Function DISPLAY-OF(National-data,1399)
      To Japanese-MBCS-data
    . . .
(5) Move Function NATIONAL-OF(Japanese-MBCS-data, 1399)
      To National-data
(6) Display National-data
    . . .
    Goback.
```

This program requires conversion tables for the following CCSID pairs:

- CCSID 1140 to CCSID 1200 (the CCSID in effect for the CODEPAGE compiler option to 1200)
- CCSID 1200 to CCSID 1140 (1200 to the CCSID in effect for the CODEPAGE compiler option)
- CCSID 1200 to CCSID 1252 (required for the DISPLAY-OF function in statement 3 above)
- CCSID 1200 to CCSID 1399 (required for the DISPLAY-OF function in statement 4 above)
- CCSID 1399 to CCSID 1200 (required for the NATIONAL-OF function in statement 5 above)

Considerations for COBOL DB2 programs

When a COBOL program contains DB2 SQL statements, you need to configure Unicode conversion services with the conversion tables that DB2 requires as well as those that COBOL requires. One key difference for DB2 is that DB2 uses the conversion technique search order ER rather than RECLM. Therefore, even if COBOL and DB2 require conversions between the same CCSID pair, you must create two conversion tables for the pair: one for COBOL with the technique search order RECLM (or omit the technique search order because it is the default) and the other with the technique search order ER.

For additional information about DB2 requirements for configuring Unicode conversion services, see:

- The DB2 library at:
  http://www.ibm.com/software/data/db2/os390/library.html
- *DB2 for OS/390 and z/OS Version 7 Installation Guide* at:
  http://www.ibm.com/support/manager.wss?rs=0&rt=0&org=SW&doc=7001469

Example: JCL for generating a conversion image

The following sample batch job supports the following three types of COBOL programs:

- A COBOL program that contains the language elements shown above
- A COBOL program that contains OO syntax for Java interoperability
- A COBOL DB2 program that requires conversions between CCSID 1200 and CCSID 1140 by DB2

This sample JCL is a variation of the *hlq*.SCUNJCL(CUNJIUTL) that is provided in the Unicode conversion services package.

```
//CUNMIUTL EXEC PGM=CUNMIUTL
//SYSPRINT DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
//SYSIMG   DD   DSN=UNI.IMAGES(CUNIMG01),DISP=SHR
//TABIN    DD   DSN=UNI.SCUNTBL,DISP=SHR
//SYSIN    DD   *
  /****************************************************************/
  /* Conversion image input for conversions between national      */
  /* data (1200) and alphanumeric, DBCS or MBCS data with         */
  /* CCSID in effect for CODEPAGE compiler option                 */
  /* Default "technique search order" (RECLM) is used by COBOL    */
  /****************************************************************/
  CONVERSION 1140,1200; /* Latin-1 to UTF-16, RECLM              */
  CONVERSION 1200,1140; /* UTF-16 to Latin-1, RECLM              */

  /****************************************************************/
  /* Conversion image input for CCSIDs specified                  */
  /* in DISPLAY-OF and NATIONAL-OF intrinsic functions            */
  /* DISPLAY-OF requires conversion from 1200 to specified CCSID  */
  /* NATIONAL-OF requires conversion from specified CCSID to 1200 */
  /* Default "technique search order" (RECLM) is used by COBOL    */
  /****************************************************************/
  CONVERSION 1200,1252; /* UTF-16 to ASCII Latin-1, RECLM        */
  CONVERSION 1200,1399; /* UTF-16 to Japanese, RECLM             */
  CONVERSION 1399,1200; /* Japanese to UTF-16, RECLM             */

  /****************************************************************/
  /* Conversion image input for COBOL OO support                  */
  /* Default "technique search order" (RECLM) is used by COBOL    */
  /****************************************************************/
  CONVERSION 1140,1208; /* Latin-1 to UTF-8, RECLM               */
  CONVERSION 1200,1208; /* UTF-16 to UTF-8, RECLM                */

  /****************************************************************/
  /* Conversion image input for DB2                               */
  /* "technique search order" used by DB2 is ER                   */
  /****************************************************************/
  CONVERSION 1140,1200,ER; /* Latin-1 to UTF-16, ER              */
  CONVERSION 1200,1140,ER; /* UTF-16 to Latin-1, ER              */
/*
```

COBOL uses the character conversion services but not the case conversion or the normalization services of Unicode conversion services. You do not need to include

CASE or NORMALIZE control statements for the conversion image creation unless other products or applications require them.

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4

555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Programming interface information

Enterprise COBOL for z/OS and OS/390 provides no macros that allow a customer installation to write programs that use the services of Enterprise COBOL for z/OS and OS/390.

Attention: Do not use as programming interfaces any Enterprise COBOL for z/OS and OS/390 macros.

Trademarks

List of resources

Enterprise COBOL for z/OS and OS/390

Migration Guide, GC27-1409

Customization Guide, GC27-1410

Debug Tool User's Guide, SC18-7171

Debug Tool Reference and Messages, SC18-7172

Fact Sheet, GC27-1407

Language Reference, SC27-1408

Licensed Product Specifications, GC27-1411

Programming Guide, SC27-1412

Language Environment for OS/390 & VM

OS/390 Language Environment Concepts Guide, GC28-1945

OS/390 Language Environment Debugging Guide, SC28-1942

OS/390 Language Environment Customization for OS/390, SC28-1941

OS/390 Language Environment Programming Guide, SC28-1939

OS/390 Language Environment Programming Reference, SC28-1940

OS/390 Run-Time Migration Guide, SC28-1944

OS/390 Writing Interlanguage Communications Applications, SC28-1943

z/OS Language Environment

z/OS Language Environment Concepts Guide, SA22-7567

z/OS Language Environment Debugging Guide, GA22-7560

z/OS Language Environment Run-Time Messages, SA22-7566

z/OS Language Environment Customization, SA22-7564

z/OS Language Environment Programming Guide, SA22-7561

z/OS Language Environment Programming Reference, SA22-7562

z/OS Language Environment Run-Time Migration Guide, GA22-7565

z/OS Language Environment Writing Interlanguage Communication Applications, SA22-7563

Related publications

American National Standard ANSI INCITS 23-1985, Programming languages - COBOL, as amended by ANSI INCITS 23a-1989, Programming Languages - Intrinsic Function Module for COBOL, and ANSI INCITS 23b-1993, Programming Languages - Correction Amendment for COBOL

CICS Transaction Server Application Programming Guide, SC33-1687

CICS Transaction Server Application Programming Reference, SC33-1688

CICS Transaction Server Customization Guide, SC33-1683

CICS Transaction Server External Interfaces Guide, SC33-1944

DBCS Ordering Support Program (DBCSOS): Program Description, N:SH18-0144 (in Japanese)

International Standard ISO 1989:1985, Programming languages - COBOL, as amended by ISO/IEC 1989/AMD1:1992, Programming languages - COBOL - Intrinsic function module, and ISO/IEC 1989/AMD2:1994, Programming languages - Correction and clarification amendment for COBOL

OS/390 MVS Initialization and Tuning Guide, SC28-1751

z/OS MVS Initialization and Tuning Guide, SA22-7591

OS/390 MVS Initialization and Tuning Reference, SC28-1752

z/OS MVS Initialization and Tuning Reference, SA22-7592

z/OS *Support for Unicode: Using Conversion Services*, SA22-7649

OS/390 SMP/E User's Guide, SC28-1740

OS/390 SMP/E Reference, SC28-1806

SMP/E Release 8.1 User's Guide, SC28-1302

Support for Unicode using Conversion Services, SC33-7050

SMP/E Release 8.1 Reference, SC28-1107

Softcopy publications

The following collection kits contains IBM COBOL and related product publications.

- *OS/390 collection*, SK2T-6700
- *z/OS collection*, SK3T-4269

Index

Readers' Comments — We'd Like to Hear from You

Enterprise COBOL for z/OS and OS/390
Customization Guide
Version 3 Release 2

Publication No. GC27-1410-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
GC27-1410-02

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
555 Bailey Avenue H150/090
San Jose, CA
U.S.A.
95141-9989

Fold and Tape **Please do not staple** Fold and Tape

GC27-1410-02

Cut or Fold
Along Line

Program Number: 5655-G53

Printed in U.S.A.

GC27-1410-02