

Enterprise COBOL for z/OS
6.4

Customization Guide



Note

Before using this information and the product it supports, be sure to read the general information under [“Notices” on page 89](#).

Fourth edition (30 April 2024 update)

This edition applies to Version 6 Release 4 of IBM® Enterprise COBOL for z/OS® (program number 5655-EC6) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure that you are using the correct edition for the level of the product.

You can view or download softcopy publications free of charge in the [Enterprise COBOL for z/OS library](#). Because Enterprise COBOL for z/OS supports the continuous delivery (CD) model and publications are updated to document the features delivered under the CD model, it is a good idea to check for updates once every two months.

It is our intention to update the product documentation for this release periodically, without updating the order number. If you need to uniquely refer to the version of your product documentation, refer to the order number with the date of update.

© **Copyright International Business Machines Corporation 1996, 2024.**

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

Contents

Figures.....	vii
Tables.....	ix
Preface.....	xi
About this information.....	xi
How to read the syntax diagrams.....	xi
How to use examples.....	xii
Using the macro planning worksheets.....	xii
Summary of changes.....	xiii
Enterprise COBOL for z/OS 6.4.....	xiii
How to send your comments.....	xiii
Chapter 1. Planning to customize Enterprise COBOL.....	1
Making changes after installation: why customize?.....	1
Planning to modify compiler option default values.....	1
Modifying compiler options	2
Sample installation jobs to modify IGYCDOPT compiler option defaults.....	5
Overriding compiler options that are fixed.....	7
Planning to create an additional reserved word table.....	7
Why create additional reserved word tables?.....	7
Controlling use of nested programs.....	7
Reserved word tables supplied with Enterprise COBOL.....	8
Using product registration to enable or disable Enterprise COBOL.....	9
Chapter 2. Enterprise COBOL compiler options.....	11
Specifying COBOL compiler options.....	11
Conflicting compiler options.....	11
Compiler options for standards conformance.....	13
Compiler options syntax and descriptions.....	13
ADATA.....	13
ADEXIT.....	13
ADV.....	14
AFP.....	14
ALLOWCBL.....	15
ALLOWCOPYLOC.....	15
ALLOWDEFINE	16
ARCH.....	16
ARITH.....	18
AWO.....	18
BLOCK0.....	19
BUF.....	19
CICS.....	20
CODEPAGE.....	20
COMPILE.....	21
CONDCOMP.....	21
COPYRIGHT.....	22
CURRENCY.....	22
DATA.....	23

DBCS.....	24
DBCSXREF.....	24
DECK.....	25
DIAGTRUNC.....	25
DISPSIGN.....	26
DLL.....	27
DYNAM.....	27
EXPORTALL.....	28
FASTSRT.....	28
FLAG.....	29
FLAGSTD.....	30
HGPR.....	32
INEXIT.....	32
INITCHECK.....	33
INITIAL.....	34
INLINE.....	34
INTDATE.....	35
INVDATA.....	35
LANGUAGE.....	38
LIBEXIT.....	39
LINECNT.....	40
LIST.....	40
LITCHAR.....	41
LP.....	41
MAP.....	42
MAXPCF.....	42
MDECK.....	43
MSGEXIT.....	44
NAME.....	44
NSYMBOL.....	44
NUM.....	45
NUMCHECK.....	45
NUMCLS.....	49
NUMPROC.....	50
OBJECT.....	50
OFFSET.....	51
OPTIMIZE.....	51
OUTDD.....	52
PARMCHECK.....	52
PGMNAME.....	53
PRTEXTIT.....	54
QUALIFY.....	54
RENT.....	55
RMODE.....	56
RULES.....	57
SEQ.....	58
SERVICE.....	59
SMARTBIN.....	59
SOURCE.....	60
SPACE.....	60
SQL.....	61
SQLCCSID.....	61
SQLIMS.....	62
SSRANGE.....	63
STGOPT.....	64
SUPPRESS.....	64
TERM.....	65
TEST.....	65

THREAD.....	67
TRUNC.....	69
TUNE.....	70
VBREF.....	71
VLR.....	71
VSAMOPENFS.....	72
WORD.....	73
XMLPARSE.....	74
XREFOPT.....	74
ZONECHECK.....	75
ZONEDATA.....	75
ZWB.....	77

Chapter 3. Customizing Enterprise COBOL..... 79

Summary of user modifications.....	79
Changing the defaults for compiler options.....	79
Changing the compiler options default module.....	80
Overriding options specified as fixed.....	81
Changing reserved words.....	81
Creating or modifying a reserved word table.....	82
Coding control statements.....	83
Rules for coding control statements.....	83
Coding operands in control statements.....	84
Rules for coding control statement operands.....	84
ABBR statement.....	84
INFO statement.....	84
RSTR statement.....	85
Modifying and running non-SMP/E JCL.....	85
Running JCL that creates a reserved word table.....	86
Tailoring the cataloged procedures to your site.....	86

Appendix A. Accessibility features for Enterprise COBOL for z/OS..... 87

Notices..... 89

Programming interface information.....	91
Trademarks.....	91

Glossary..... 93

List of resources..... 137

Enterprise COBOL for z/OS.....	137
Related publications.....	137

Index..... 141

Figures

1. Syntax format for IGYCOPT compiler options macro..... 2

2. Syntax format for reserved word control statements..... 83

Tables

1. IGYCDOPT worksheet for options.....	2
2. Conflicting compiler options.....	11
3. DISPLAY output with the DISPSIGN=COMPAT option or the DISPSIGN=SEP option specified:.....	26
4. Setting INVDATA and NUMPROC options when migrating from earlier COBOL versions.....	36
5. Entries for the LANGUAGE compiler option.....	39
6. Effect of RENT and RMODE on residency mode.....	55
7. Effect of RMODE and RENT NORENT on residency mode.....	56
8. Length of record read and file status.....	72
9. Summary of user modification jobs for Enterprise COBOL.....	79

Preface

About this information

This information is intended for systems programmers who are responsible for customizing Enterprise COBOL for their location. It provides information needed to plan for and customize Enterprise COBOL under z/OS. This information can also help you assess the value of Enterprise COBOL to your organization.

Throughout this information, "COBOL" or "Enterprise COBOL" refers to "IBM Enterprise COBOL for z/OS" or "IBM Enterprise COBOL Value Unit Edition for z/OS". The generic term "operating system" refers to z/OS.

To use this information, and ensure successful customization, you should have a knowledge of Enterprise COBOL and of your system's operating environment.

How to read the syntax diagrams

This section describes how to read the syntax diagrams in this information.

- 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.

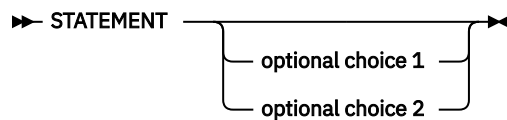
▶▶ STATEMENT — optional item ▶▶

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

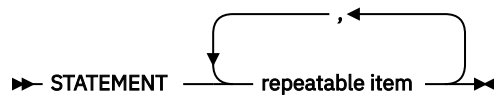
If you *must* choose one of the items, one item of the stack appears on the main path. The default, if any, appears above the main path and is chosen by the IGYCOPT macro if you do not specify another choice. In some cases, the default is affected by the system in which the program is being run.

▶▶ STATEMENT — default-item — required choice 1 — required choice 2 — ▶▶

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



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



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 an italic font (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.

For a description of the meaning of the asterisk (*) in syntax diagrams, and for further information, see [“Compiler options syntax and descriptions” on page 13.](#)

How to use examples

This information shows numerous examples of sample COBOL statements, program fragments, and small programs to illustrate the coding techniques being described. The examples of program code are written in lowercase, uppercase, or mixed case to demonstrate that you can write your programs in any of these ways.

To more clearly separate some examples from the explanatory text, they are presented in a monospace font.

COBOL keywords and compiler options that appear in text are generally shown in SMALL UPPERCASE. Other terms such as program variable names are sometimes shown in *an italic font* for clarity.

If you copy and paste examples from the PDF format documentation, make sure that the spaces in the examples (if any) are in place; you might need to manually add some missing spaces to ensure that COBOL source text aligns to the required columns per the COBOL reference format in the *Enterprise COBOL Language Reference*. Alternatively, you can copy and paste examples from the HTML format documentation and the spaces should be already in place.

Using the macro planning worksheets

The planning worksheets in this information ([“IGYCDOPT worksheet for compiler options” on page 2](#)) will help you prepare to customize Enterprise COBOL. By completing the worksheets, you will be able to easily identify those values that you want to change from the IBM-supplied defaults. You might then want to use the worksheets as a source from which to customize the IBM-supplied default values.

The headings in each worksheet differ somewhat from each other. See the following list of definitions for an explanation of the column headings in the worksheet for compiler options.

Compiler option

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

Enter * for fixed

The options that cannot be overridden by an application programmer. Enter an asterisk (*) only for those options that you want to be fixed.

Enter selection

The value associated with each option. In the space provided, enter the value that you want to assign to each option. To assist you in selecting the appropriate value, see the reference in the **Syntax description** column.

IBM-supplied default

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

Syntax description

The topic that contains the syntax diagram and more specific information about the given 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 are positioned such that the order of the entries is consistent with the actual coding semantics.

Summary of changes

This section lists the key changes that have been made to this document since Enterprise COBOL for z/OS 6.4. The latest technical changes are marked within >| and <| in the HTML version, or marked by vertical bars (|) in the left margin in the PDF version.

For a complete list of new and improved features in Enterprise COBOL for z/OS 6.4 and COBOL 6.4 with PTFs installed, see What is new in Enterprise COBOL for z/OS 6.4 and COBOL 6.4 with PTFs installed in the *Enterprise COBOL for z/OS What's New*.

Enterprise COBOL for z/OS 6.4

Compiler option changes

- The following compiler option is new:
 - SMARTBIN: Use SMARTBIN to instruct the compiler to generate modules containing additional binary metadata that enables them to be optimized by IBM Automatic Binary Optimizer (ABO) for z/OS 2.2. (“SMARTBIN” on page 59)
 - PH50296: CONDCOMP: The new CONDCOMP option is introduced to control how conditional code will be displayed in the listing. (CONDCOMP)
- The following compiler options are modified:
 - ARCH: ARCH=8 and ARCH=9 are no longer accepted. A new higher level of ARCH=14 is accepted. ARCH=10 is the default. (“ARCH” on page 16)
 - TUNE: TUNE=8 and TUNE=9 are no longer accepted. A new higher level of TUNE=14 is accepted. TUNE=10 is the default if ARCH is not specified. (“TUNE” on page 70)

How to send your comments

Your feedback is important in helping us to provide accurate, high-quality information. If you have comments about this information or any other Enterprise COBOL documentation, send your comments to: compinfo@cn.ibm.com.

Be sure to include the name of the document, the publication number, the version of Enterprise COBOL, and, if applicable, the specific location (for example, the page number or section heading) 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 that IBM believes appropriate without incurring any obligation to you.

Chapter 1. Planning to customize Enterprise COBOL

When you plan the customization of Enterprise COBOL, you need to consider whether to modify compiler-option default values and whether to create an additional reserved-word table.

The following information helps you plan your customization:

- [“Making changes after installation: why customize?” on page 1](#)
- [“Planning to modify compiler option default values” on page 1](#)
- [“Planning to create an additional reserved word table” on page 7](#)
- [“Using product registration to enable or disable Enterprise COBOL” on page 9](#)

If you're installing z/OS Debugger, you can decide whether to place its modules in shared storage, and whether to set up your CICS® environment to work with the debugger.

For the actual customization procedures, see [Chapter 3, “Customizing Enterprise COBOL,” on page 79](#).

This information also contains worksheets to help you plan modifications to the IBM-supplied default values within macros. For an explanation about the planning sheets, see [“Using the macro planning worksheets” on page xii](#).

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 for 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:

- Modify the default values of compiler options: see [“Planning to modify compiler option default values” on page 1](#).
- Make compiler options fixed: see [“Overriding compiler options that are fixed” on page 7](#).
- Create additional reserved word tables: see [“Planning to create an additional reserved word table” on page 7](#).

Planning to modify compiler option default values

IBM provides a default setting for each compiler option. You can accept the IBM-supplied compiler option values 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 your application programmers will have the ability to override an option default (meaning the option default is not fixed) or will be restricted from overriding an option default (meaning the option default is fixed). The IGYCDOPT program lets you set and fix the defaults for the compiler options.

Compiler option defaults is set in the IGYCDOPT program.

IGYCDOPT is link-edited with AMODE 31 and RMODE ANY during installation.

For compiler option defaults set in the IGYCDOPT program, see [Table 1 on page 2](#).

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.V6R4M0.SIGYMAC.

Note: The high-level qualifier IGY.V6R4M0 might have been changed when Enterprise COBOL was installed.

Modifying compiler options

IBM provides a default setting for each compiler option which can be modified to better suit your needs at your location.

The compiler options and their defaults are described in [“IGYCDOPT worksheet for compiler options” on page 2](#). Review these options and their default values to determine the values that are most suitable for your applications.

If you plan to modify the values for compiler options, use the following IGYCOPT syntax format.

Note: Prefacing an option's value with an asterisk will set that option's value as "fixed", meaning that it cannot be overridden at compile time, unless special steps are taken to do so. See [“Overriding compiler options that are fixed” on page 7](#) for more information on how options whose defaults are set as fixed can be overridden later at compiler time if needed.

IGYCOPT format

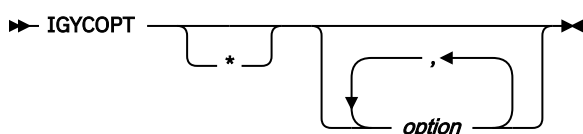


Figure 1. Syntax format for IGYCOPT compiler options macro

IGYCDOPT worksheet for compiler options

The IGYCDOPT worksheet will help you plan and set the compiler options portion in the IGYCDOPT program. The IBM-supplied default values are shown in the "IBM-supplied default" column. The default is also visible following the syntax diagram by clicking on the option in the "Syntax description" column. For more information about the syntax diagrams, see [“How to read the syntax diagrams” on page xi](#).

To complete the worksheet, fill in the "Enter * for fixed" and the "Enter selection" columns.

Note:

- 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. An attempt to override a fixed option at compile time will result in a diagnostic message with a nonzero compiler return code. For special circumstances where a compiler option value that you chose to set as fixed needs to be overridden by an application programmer, see [“Overriding compiler options that are fixed” on page 7](#).
- The ALLOWCBL, DBCSXREF, and NUMCLS options cannot be overridden at compile time. Therefore, the "Enter * for fixed" worksheet entries for these options are blank.
- The IBM-supplied default value for ADEXIT, INEXIT, LIBEXIT, MSGEXIT, and PRTEXTIT is null. Therefore, the "IBM-supplied default" entries for these options are blank.
- The DUMP compiler option cannot be set through the IGYCDOPT program. Unless changed at compile time, DUMP is always set to NODUMP.
- The OPTFILE compiler option cannot be set through the IGYCDOPT program.

Table 1. IGYCDOPT worksheet for options				
Compiler option	Enter * for fixed	Enter selection	IBM-supplied default	Syntax description
ADATA=	----	-----	NO	“ADATA” on page 13
ADEXIT=	----	-----		“ADEXIT” on page 13

Table 1. IGYCDOPT worksheet for options (continued)

Compiler option	Enter * for fixed	Enter selection	IBM-supplied default	Syntax description
ADV=	----	-----	<u>YES</u>	“ADV” on page 14
AFP=	----	-----	<u>NOVOLATILE</u>	“AFP” on page 14
ALLOWCBL=	----	-----	<u>YES</u>	“ALLOWCBL” on page 15
ALLOWCOPYLOC=	----	-----	<u>YES</u>	“ALLOWCOPYLOC” on page 15
ALLOWDEFINE=	----	-----	<u>YES</u>	“ALLOWDEFINE ” on page 16
ARCH=	----	-----	<u>10</u>	“ARCH” on page 16
ARITH=	----	-----	<u>COMPAT</u>	“ARITH” on page 18
AWO=	----	-----	<u>NO</u>	“AWO” on page 18
BLOCK0=	----	-----	<u>NO</u>	“BLOCK0” on page 19
BUF=	----	-----	<u>4K</u>	“BUF” on page 19
CICS=	----	-----	<u>NO</u>	“CICS” on page 20
CODEPAGE=	----	-----	<u>1140</u>	“CODEPAGE” on page 20
COMPILE=	----	-----	<u>NOC(S)</u>	“COMPILE” on page 21
CONDCOMP=	----	-----	<u>NOSKIPSRC</u>	“CONDCOMP” on page 21
COPYRIGHT=	----	-----	<u>NO</u>	“COPYRIGHT” on page 22
CURRENCY=	----	-----	<u>NO</u>	“CURRENCY” on page 22
DATA=	----	-----	<u>31</u>	“DATA” on page 23
DBCS=	----	-----	<u>Yes</u>	“DBCS” on page 24
DBCSXREF=	----	-----	<u>NO</u>	“DBCSXREF” on page 24
DECK=	----	-----	<u>NO</u>	“DECK” on page 25
DIAGTRUNC=	----	-----	<u>NO</u>	“DIAGTRUNC” on page 25
DISPSIGN=	----	-----	<u>COMPAT</u>	“DISPSIGN” on page 26
DLL=	----	-----	<u>NO</u>	“DLL” on page 27
DYNAM=	----	-----	<u>NO</u>	“DYNAM” on page 27
EXPORTALL=	----	-----	<u>NO</u>	“EXPORTALL” on page 28
FASTSRT=	----	-----	<u>NO</u>	“FASTSRT” on page 28
FLAG=	----	-----	<u>(I,I)</u>	“FLAG” on page 29
FLAGSTD=	----	-----	<u>NO</u>	“FLAGSTD” on page 30
HGPR=	----	-----	<u>PRESERVE</u>	“HGPR” on page 32
INEXIT=	----	-----		“INEXIT” on page 32
INITCHECK=	----	-----	<u>NO</u>	“INITCHECK” on page 33
INITIAL=	----	-----	<u>NO</u>	“INITIAL” on page 34
INLINE=	----	-----	<u>YES</u>	“INLINE” on page 34

Table 1. IGYCDOPT worksheet for options (continued)

Compiler option	Enter * for fixed	Enter selection	IBM-supplied default	Syntax description
INTDATE=	----	-----	<u>ANSI</u>	“INTDATE” on page 35
INVDATA=	----	-----	<u>NO</u>	“INVDATA” on page 35
LANGUAGE=	----	-----	<u>EN</u>	“LANGUAGE” on page 38
LIBEXIT=	----	-----		“LIBEXIT” on page 39
LINECNT=	----	-----	<u>60</u>	“LINECNT” on page 40
LIST=	----	-----	<u>NO</u>	“LIST” on page 40
LITCHAR=	----	-----	<u>QUOTE</u>	“LITCHAR” on page 41
LP=	----	-----	<u>32</u>	LP
MAP=	----	-----	<u>NO</u>	“MAP” on page 42
MAXPCF=	----	-----	<u>100000</u>	“MAXPCF” on page 42
MDECK=	----	-----	<u>NO</u>	“MDECK” on page 43
MSGEXIT=	----	-----		“MSGEXIT” on page 44
NAME=	----	-----	<u>NO</u>	“NAME” on page 44
NSYMBOL=	----	-----	<u>NATIONAL</u>	“NSYMBOL” on page 44
NUM=	----	-----	<u>NO</u>	“NUM” on page 45
NUMCHECK=	----	-----	<u>(NO)</u>	“NUMCHECK” on page 45
NUMCLS=	----	-----	<u>PRIM</u>	“NUMCLS” on page 49
NUMPROC=	----	-----	<u>NOPFD</u>	“NUMPROC” on page 50
OBJECT=	----	-----	<u>YES</u>	“OBJECT” on page 50
OFFSET=	----	-----	<u>NO</u>	“OFFSET” on page 51
OPTIMIZE=	----	-----	<u>0</u>	“OPTIMIZE” on page 51
OUTDD=	----	-----	<u>SYSOUT</u>	“OUTDD” on page 52
PARMCHECK=	----	-----	<u>(NO)</u>	“PARMCHECK” on page 52
PGMNAME=	----	-----	<u>COMPAT</u>	“PGMNAME” on page 53
PRTEXIT=	----	-----		“PRTEXIT” on page 54
QUALIFY=	----	-----	<u>COMPAT</u>	“QUALIFY” on page 54
RENT=	----	-----	<u>YES</u>	“RENT” on page 55
RMODE=	----	-----	<u>AUTO</u>	“RMODE” on page 56
RULES=	----	-----	<u>(NO)</u>	“RULES” on page 57
SEQ=	----	-----	<u>YES</u>	“SEQ” on page 58
SERVICE=	----	-----	<u>NO</u>	“SERVICE” on page 59
SMARTBIN=	----	-----	YES when LP=32 is in effect. SMARTBIN is not supported when LP=64 is in effect.	“SMARTBIN” on page 59

Table 1. IGYCDOPT worksheet for options (continued)

Compiler option	Enter * for fixed	Enter selection	IBM-supplied default	Syntax description
SOURCE=	----	-----	<u>YES</u>	“SOURCE” on page 60
SPACE=	----	-----	<u>1</u>	“SPACE” on page 60
SQL=	----	-----	<u>NO</u>	“SQL” on page 61
SQLCCSID=	----	-----	<u>YES</u>	“SQLCCSID” on page 61
SQLIMS=	----	-----	<u>NO</u>	“SQLIMS” on page 62
SSRANGE=	----	-----	<u>NO</u>	“SSRANGE” on page 63
STGOPT=	----	-----	<u>NO</u>	“STGOPT” on page 64
SUPPRESS=	----	-----	<u>YES</u>	“SUPPRESS” on page 64
TERM=	----	-----	<u>NO</u>	“TERM” on page 65
TEST=	----	-----	<u>(NO, NODWARF)</u>	“TEST” on page 65
THREAD=	----	-----	<u>NO</u>	“THREAD” on page 67
TRUNC=	----	-----	<u>STD</u>	“TRUNC” on page 69
TUNE=	----	-----	<u>10 if ARCH is not specified. The default TUNE level matches the ARCH level if ARCH is specified.</u>	“TUNE” on page 70
VBREF=	----	-----	<u>NO</u>	“VBREF” on page 71
VLR=	----	-----	<u>STANDARD</u>	“VLR” on page 71
VSAMOPENFS=	----	-----	<u>COMPAT</u>	“VSAMOPENFS” on page 72
WORD=	----	-----	<u>NO</u>	“WORD” on page 73
XMLPARSE=	----	-----	<u>XMLSS</u>	“XMLPARSE” on page 74
XREFOPT=	----	-----	<u>FULL</u>	“XREFOPT” on page 74
ZWB=	----	-----	<u>YES</u>	“ZWB” on page 77

Sample installation jobs to modify IGYCDOPT compiler option defaults

Enterprise COBOL provides two sample installation jobs that you can modify and then use to change the defaults for compiler options. The first sample job, IGYWDOPT, provides an example of how to change the IBM-supplied defaults for compilers. This job then uses SMP/E to install the newly built compiler option default module, IGYCDOPT, into SIGYCOMP. The second sample job, IGYWUOPT, provides an example of how to override compiler options that have been fixed. This job will place the newly built compiler option default module, IGYCDOPT, into an application execution data set, that can then be included in the application's STEPLIB DD in the compile JCL. These jobs are located in the COBOL sample data set IGY.V6R4M0.SIGYSAMP.

IGYWDOPT

This sample installation job can be used to change the IBM-supplied compiler option defaults. It will create a new IGYCDOPT module and then apply it into SIGYCOMP using SMP/E. If SMP/E is used to

install the COBOL compiler, then use this sample installation job to change the IBM-supplied compiler option defaults.

If IGYWDOPT is being run for the first time, complete the following steps:

1. Change the job card to meet your system requirements.
2. Change these items:
 - #globalcsi (Make it the CSI name of the installation site)
 - #tzone (Make it the TARGET ZONE name of the installation site)
3. Copy member SIGYSAMP(IGYCDOPT) into SIGYSAMP(IGYWDOPT) in place of the comment lines following the ++ SRC statement in step DOPT.
4. Modify the IGYCDOPT text that was just copied in so that it contains the list of compiler options that need to be overridden. For example:

```
COPY IGYCDOPT
IGYCDOPT CSECT
IGYCDOPT AMODE ANY
IGYCDOPT RMODE ANY
          IGYCOPT  ARCH=10,                X
                   OPTIMIZE=*2,           X
                   NUMCHECK=(ZON,PAC,BIN,MSG), X
                   INVDATA=NO
          END IGYCDOPT
```

Use the continuation "X" in column 72 as needed.

5. Run IGYWDOPT to receive and apply the usermod to create a customized version of MOD(IGYCDOPT).
6. **Important:** Save a copy of the modified SIGYSAMP(IGYWDOPT) for future reference.

CAUTION: Do not ACCEPT the usermod! Accepting the usermod makes it impossible to RESTORE it later in SMP/E when needed.

If MOD(IGYCDOPT) is being changed by an IBM PTF and requires a rerun of the SIGYSAMP(IGYWDOPT) job, complete the following steps:

1. RESTORE the usermod created by the IGYWDOPT job. This is done via the SMP/E command RESTORE SELECT (IGYWDOP). Doing this will restore MOD(IGYCDOPT) back to the previous IBM PTF level, which will then allow the new IBM PTF to apply properly.
2. Apply the IBM PTF.
3. Change the rework date by changing the REWORK parameter on the ++ USERMOD statement to the date the changes are being made.
4. Add the proper "PRE()" statement after the "FMID()" statement. This is typically the PTF number that was just applied. See the [technote](#) to determine what PRE statement to add if an error occurs.
5. Using the SIGYSAMP(IGYWDOPT) backup member as a reference and referring to any options that may have been added or modified in SIGYMAC(IGYCOPT), update IGYWDOPT so that it contains the list of compiler options that need to be overridden.
6. Rerun IGYWDOPT to receive and apply the usermod to recreate a customized version of MOD(IGYCDOPT).
7. **Important:** Save off a copy of the modified SIGYSAMP(IGYWDOPT) for future reference.

IGYWDOPT should run with a condition code of 0.

Check the IGYNNNN informational messages in the ASSEMBLER SYSPRINT data set to verify the options that will be in effect when the new IGYCDOPT module is used.

IGYWUOPT

This sample installation job can be used to change the IBM-supplied compiler option defaults. If SMP/E is not used to install the COBOL compiler, which is a rare case, then this sample installation job can be used to change the IBM-supplied defaults and copy the new IGYCDOPT module into SIGYCOMP. This job can also be used to create an IGYCDOPT compiler option default module that

can be placed into an application execution data set and then referenced by the application in the STEPLIB DD in the compile JCL. In this way, compiler option defaults in SIGYCOMP (IGYCDOPT) that are fixed and cannot normally be overridden can be overridden.

Overriding compiler options that are fixed

At installation time, the IGYCDOPT program (sample jobs SIGYSAMP(IGYWDOPT & IGYWUOPT) can be used to specify that a compiler option is fixed and cannot be changed or overridden at compile time. This means that at compile time, an attempt to override the fixed option will result in a diagnostic message with a nonzero compiler return code.

However, there may be special conditions that require the ability to override a fixed option. For example, if OPT=2 is fixed at the installation level (indicating that you always want the COBOL compiler to generate optimized object code), and you have an immediate need to compile without optimization, then you would need a way to override the fixed OPT=2 option, so that OPT=0 could be used instead.

This change can be made by assembling a temporary copy of the IGYCDOPT program to contain a (non-fixed) default for the compiler option and then placing that copy of IGYCDOPT into an application's data set. At compile time, programmers can then use a JOBLIB or STEPLIB to point to the data set that contains the modified IGYCDOPT module to bypass the fixed compiler option.

The sample job, SIGYSAMP(IGYWUOPT) can be used to accomplish this task. Carefully follow the instructions in the comments in SIGYSAMP(IGYWUOPT) to tailor that sample job.

Note:

- //SYSLMOD should point to the application data set where the new IGYCDOPT module will reside.
- If SIGYSAMP(IGYWDOPT) was used at installation time to set the defaults, it is recommended to copy the compiler option overrides from that job into SIGYSAMP(IGYWUOPT) and then change the desired option to the new default. In this way, the application will retain all the installation defaults, except the one they chose to alter.

Planning to create an additional reserved word table

The following sections describe why you might want to create additional reserved word tables, explain how you can restrict the use of nested programs by modifying a reserved word table, and list the reserved word tables supplied with Enterprise COBOL.

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?

This section describes the benefits of creating additional reserved word tables.

You can create additional reserved word tables to:

- 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 words that are not supported under 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 by using the INFO and RSTR control statements. For instructions on how to make these modifications, see [“Creating or modifying a reserved word table” on page 82](#).

Reserved word tables supplied with Enterprise COBOL

Enterprise COBOL provides reserved word tables on the installation medium.

The reserved word tables are:

- Default reserved word table
- CICS reserved word table

Default reserved word table (IGYCRWT)

About the default reserved word table provided for your entire facility, see *Reserved words* in the *Enterprise COBOL Language Reference*.

CICS reserved word table (IGYCCICS)

Enterprise COBOL provides an alternate reserved word table for CICS application programs so that COBOL words that are not supported under CICS are flagged by the compiler.

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
- DELETE³
- FACTORY
- FD
- FILE¹
- FILE-CONTROL¹
- INPUT-OUTPUT¹
- INVOKE
- I-O-CONTROL
- MERGE
- METHOD
- OBJECT
- OPEN
- READ
- RERUN
- REWRITE
- SD^{1, 2}
- SELF
- START
- SUPER
- WRITE

Notes:

1. If you intend to use the SORT statement under CICS (COBOL supports an interface for the SORT statement under CICS), you must change the CICS reserved-word table to remove the words from the list of words marked as restricted.
2. The SORT keyword is not restricted, but the SD keyword is. This allows you to use the format 2 (table) sort statement but not the format 1 (file) sort statement.
3. If you restrict the DELETE keyword, you may still use the DELETE function of BASIS processing.

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.V6R4M0.SIGYSAMP.

Note: The high-level qualifier IGY.V6R4M0 might have been changed when Enterprise COBOL was installed.

Using product registration to enable or disable Enterprise COBOL

The default behavior for Enterprise COBOL 6 is to run on every z/OS system, but you can use the IFAPRDxx member of SYSx.PARMLIB to disable COBOL 6 from running on selected z/OS systems.

If you want to disable COBOL 6, add the following code to the active IFAPRDxx member:

```
PRODUCT OWNER('IBM CORP')
        NAME('ENTERPRISE COBOL')
        ID(5655-EC6)
        VERSION(06) RELEASE(*) MOD(*)
        FEATURENAME(*)
        STATE(DISABLED)
```

In this case, the compiler stops with RC=16 and a write-to-operator message.

If you want to explicitly enable COBOL 6, add the following code to the active IFAPRDxx member:

```
PRODUCT OWNER('IBM CORP')
        NAME('ENTERPRISE COBOL')
        ID(5655-EC6)
        VERSION(06) RELEASE(*) MOD(*)
        FEATURENAME(*)
        STATE(ENABLED)
```

However, because COBOL 6 is enabled by default, you don't have to explicitly enable COBOL 6 by adding the previous statements to the active IFAPRDxx member.

Chapter 2. Enterprise COBOL compiler options

This information describes the compiler options whose default values can be changed.

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 about the compiler options, see *Compiler options* in the *Enterprise COBOL 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 that are being developed.

Specifying COBOL compiler options

When you specify compiler options in the IGYCOPT macro, be sure to follow the assembler syntax described in the subsequent sections. This assembler syntax is not the same as the option syntax used with the compiler itself.

Always assemble your IGYCOPT macro source with the *.SIGYMAC library of the compiler for which you are preparing the default options. This will ensure that the resulting options module is consistent with the expectations of the compiler, for example when options are added or withdrawn.

Conflicting compiler options

If you specify certain compiler option values, a conflict with other compiler options might result. This topic describes possible conflicts between compiler options.

Table 2. Conflicting compiler options	
Compiler option	Conflicts with:
AFP=VOLATILE	LP=64
CICS=YES	DYNAM=YES LP=64 RENT=NO
DATA=24	LP=64
DBCS=NO	NSYMBOL=NATIONAL
DBCSXREF=(other than NO)	XREFOPT=NO
DLL=NO	EXPORTALL=YES
DLL=YES	DYNAM=YES RENT=NO

Table 2. Conflicting compiler options (continued)	
Compiler option	Conflicts with:
DYNAM=YES	CICS=YES DLL=YES EXPORTALL=YES
EXPORTALL=YES	DLL=NO DYNAM=YES RENT=NO
FLAGSTD=(other than NO)	WORD=xxxx
HGPR=NOPRESERVE	LP=64
LIST=YES	OFFSET=YES
LP=64	AFP=VOLATILE CICS=YES DATA=24 HGPR=NOPRESERVE RENT=NO RMODE=24 SMARTBIN=YES SQLIMS=YES THREAD=YES
NSYMBOL=NATIONAL	DBCS=NO
OBJECT=NO	TEST=(other than NO)
OFFSET=YES	LIST=YES
RENT=NO	CICS=YES DLL=YES EXPORTALL=YES LP=64 THREAD=YES
RMODE=24	LP=64
SMARTBIN=YES	LP=64
SQLIMS=YES	LP=64
THREAD=YES	INITIAL=YES LP=64 RENT=NO
WORD=xxxx	FLAGSTD=(other than NO)
XREFOPT=NO	DBCSXREF=(other than NO)

Compiler options for standards conformance

Several compiler options are required to conform with the 85 COBOL Standard.

For details, see *Option settings for 85 COBOL Standard conformance* in the *Enterprise COBOL Programming Guide*.

Compiler options syntax and descriptions

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

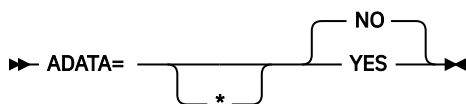
Note:

- The DUMP option is not in this list. Unless you change DUMP at compile time, it is always set to NODUMP. This option is not for general use; it is used only at the request of an IBM representative.
- The OPTFILE option is not in this list. It can be specified only as a compiler invocation PARM option, or in a PROCESS or CBL statement in the COBOL source program.
- 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

ADATA affects whether an Associated Data file is produced during compilation.

Syntax



Default

ADATA=NO

YES

Produces the Associated Data file with the appropriate records.

NO

Does not produce the Associated Data file.

Notes:

- 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.
- Selection of the Japanese language option might result in DBCS characters written records in the Associated Data file.
- Specification of NOCOMPILE(W|E|S) might stop compilation prematurely, resulting in a loss of specific Associated Data records.
- If you use the INEXIT option, the compilation source module is not identified in the SYSADATA (Associated Data file) information.

ADEXIT

ADEXIT designates a module to be called for each record that is written to the SYSADATA file.

Syntax



Default

No exit is specified. Equivalent to specifying the NOADEXIT suboption of the EXIT compiler option. If ADEXIT=* is coded without the *name* parameter, NOADEXIT cannot be overridden.

name

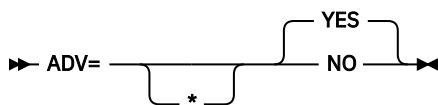
Identifies a module to be used with the EXIT compiler option. If the suboption for this user exit is specified, the compiler loads the named module and calls it for each record that is written to the SYSADATA file.

For more information about the EXIT compiler option, see *EXIT compiler option* in the *Enterprise COBOL Programming Guide*.

ADV

ADV affects WRITE ... ADVANCING statements, determining whether one byte is added to the record length for the printer control character.

Syntax



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.

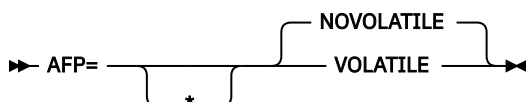
Note:

- With ADV=YES, the record length on the physical device is one byte larger than the record description length in the source program.
- If the record length for the output file is not defined in the source code, COBOL ensures that the DCB parameters are appropriately set.
- 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.
- If the LINAGE clause is specified in a file description (FD), the compiler treats that file as if ADV=YES has been specified.

AFP

The AFP option controls the compiler usage of the Additional Floating Point (AFP) registers that are provided by IBM z/Architecture processors.

Syntax



The Enterprise COBOL compiler generates code that uses the full complement of 16 floating point registers (FPR) provided by an IBM z/Architecture processor. These FPRs are as follows:

- Original FPRs, which are numbered 0, 2, 4, and 6
- AFP registers, which are numbered 1, 3, 5, 7, and 8-15

Default

AFP=NOVOLATILE

VOLATILE

If you specify AFP=VOLATILE, the AFP registers 8-15 are considered volatile, which means that they might be changed by a called subprogram. Therefore, the COBOL compiler generates extra code to protect the values in these registers.

NOVOLATILE

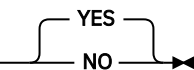
If you specify AFP=NOVOLATILE, the AFP registers 8-15 are considered nonvolatile, which means that they are known to be unchanged or preserved by every called subprogram. Therefore, the compiler can generate more efficient code sequences for programs with floating point operations. It is the normal z/OS architecture convention.

AMODE 64 considerations: When the LP=64 compiler option is in effect, the AFP=VOLATILE option is not supported. If the option is specified explicitly by the user, an informational message is issued and the setting is ignored.

ALLOWCBL

ALLOWCBL affects whether PROCESS (or CBL) statements can be used in COBOL programs.

Syntax

➔ ALLOWCBL= 

Default

ALLOWCBL=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.

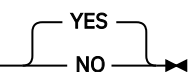
Note:

- ALLOWCBL cannot be overridden at compile time because it cannot be included in the PROCESS (or CBL) statement.
- 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 ALLOWCBL=NO.

ALLOWCOPYLOC

ALLOWCOPYLOC affects whether COPYLOC options can be used when compiling COBOL programs.

Syntax

➔ ALLOWCOPYLOC= 

Default

ALLOWCOPYLOC=YES

YES

Allows the specification of COPYLOC options when compiling COBOL programs.

NO

Diagnoses the specification of any COPYLOC options as an error.

Note:

- ALLOWCOPYLOC cannot be overridden at compile time because it cannot be included in the PROCESS (or CBL) statement.
- If your installation requirements do not allow COPYLOC compiler options to be specified in a source program, specify ALLOWCOPYLOC=NO.

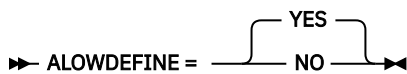
Related references

COPYLOC (*Enterprise COBOL Programming Guide*)

ALLOWDEFINE

ALLOWDEFINE affects whether DEFINE options can be used when compiling COBOL programs.

Syntax



Default

ALLOWDEFINE=YES

YES

Allows the specification of DEFINE options when compiling COBOL programs.

NO

Diagnoses the specification of any DEFINE options as an error.

Note:

- ALLOWDEFINE cannot be overridden at compile time because it cannot be included in the PROCESS (or CBL) statement.
- If your installation requirements do not allow DEFINE compiler options to be specified in a source program, specify ALLOWDEFINE=NO.

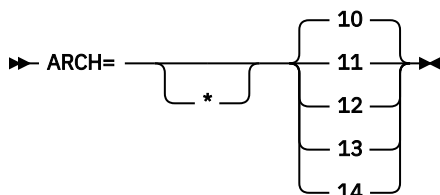
Related references

DEFINE (*Enterprise COBOL Programming Guide*)

ARCH

The ARCH option specifies the machine architecture for which the executable program instructions are to be generated.

Syntax



If you specify a higher ARCH level, the compiler generates code that uses newer and faster instructions. Your application might abend if it runs on a processor with an architecture level lower than what you specify with the ARCH option. Use the ARCH level that matches the lowest machine architecture where your application runs, including any disaster recovery (DR) machines.

Current supported architecture levels and groups of models are as follows:

Default

ARCH=10

10

Produces code that uses instructions available on the 2827-xxx (IBM zEnterprise® EC12) and 2828-xxx (IBM zEnterprise BC12) models in IBM z/Architecture mode.

Specifically, these ARCH=10 machines and their follow-ons add instructions supported by the following facilities:

- Execution-Hint Facility
- Load-and-Trap Facility
- Miscellaneous-Instructions-Extension Facility
- Transactional-Execution Facility
- Enhanced Decimal Floating Point Facility that enables more efficient conversions between zoned decimal data items and decimal floating point data items. Instead of converting zoned decimal data items to packed decimal data items to perform arithmetic when conditions permit it and the optimization level is greater than 0, the compiler converts zoned decimal data items directly to decimal floating point data items, and then back again to zoned decimal data items after the computations are complete.

11

Produces code that uses instructions available on 2964-xxx (IBM z13®) and 2965-xxx (IBM z13s®) models in IBM z/Architecture mode.

Specifically, these ARCH=11 machines and their follow-ons add instructions with support of the following facilities:

- Enhanced Decimal Floating Point Facility that enables more efficient conversions between packed-decimal data items and decimal floating point intermediate result data items when the surrounding conditions are optimal and the optimization level is greater than 0.
- Exploitation of the Vector Extension Facility (SIMD) instructions for some INSPECT REPLACING and INSPECT TALLYING statements.

To use the Vector Extension Facility (SIMD) instructions, the code must be executed on a machine running on z/OS 2.2, or z/OS 2.1 with the PTFs for APARs OA43803 and PI12412 installed.

12

Produces code that uses instructions available on 3906-xxx (IBM z14) and 3907-xxx (IBM z14 ZR1) models in IBM z/Architecture mode.

Specifically, these ARCH=12 machines and their follow-ons add instructions supported by the Vector Packed Decimal Facility, which accelerates packed and zoned decimal computation by storing intermediate results in vector registers instead of in memory.

13

Produces code that uses instructions available on the 8561-xxx (IBM z15) and 8562-xxx (IBM z15 T02) models in IBM z/Architecture mode.

Specifically, these ARCH=13 machines and their follow-ons add instructions supported by the following facilities:

- Vector Packed Decimal Enhancement Facility
- Vector-Enhancements Facility 2
- Miscellaneous Instruction-Extensions-Facility 3
- Aligned vector load/store hints

14

Produces code that uses instructions available on the 3931-xxxx (IBM z16) model in IBM z/Architecture mode.

Specifically, this ARCH=14 machine and its follow-ons add instructions supported by the new Vector Packed Decimal Enhancement Facility 2. This new facility adds performance improvements for COBOL programs that contain one or more of the following types of statements:

- Exponentiation operations on packed or zoned decimal data items where the exponent is declared with one or more fractional digits
- Arithmetic statements involving mixed decimal and floating-point data items
- Statements using numeric-edited data items

Note: A higher ARCH level includes the facilities of the lower ARCH levels. For example, ARCH=14 includes all the facilities of the lower ARCH levels.

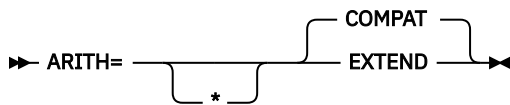
For more information about these facilities, see *IBM z/Architecture Principles of Operation*.

Use the ARCH option together with the TUNE option. For more information about the interaction between ARCH and TUNE, see [“TUNE” on page 70](#).

ARITH

ARITH affects the maximum number of digits that can be coded for integers, and the number of digits used in fixed-point intermediate results.

Syntax



Default

ARITH=COMPAT

COMPAT

Specifies 18 digits as the maximum precision for decimal data.

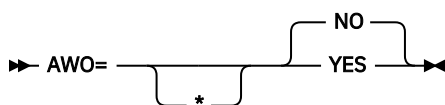
EXTEND

Specifies 31 digits as the maximum precision for decimal data.

AWO

AWO affects whether the APPLY-WRITE-ONLY clause is activated for physical-sequential files that have variable blocked format.

Syntax



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 runtime files when handling input and output.

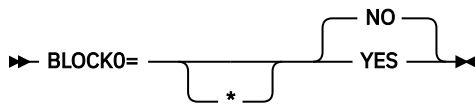
NO

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.

BLOCK0

BLOCK0 affects whether the default blocking specification for QSAM files is changed from unblocked to blocked.

Syntax



Default

BLOCK0=NO

YES

Changes the default blocking specification for QSAM files that specify neither BLOCK CONTAINS nor RECORDING MODE U in the file description entry. BLOCK0=YES activates the BLOCK CONTAINS 0 clause for such files, causing them to have a system-determined block size at run time.

Performance consideration: Using BLOCK0=YES could result in enhanced processing speed and minimized storage requirements for QSAM output files. But see the recommendation below.

NO

Does not activate the BLOCK CONTAINS 0 clause by default for any file.

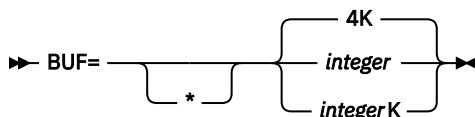
Recommendation: Adding a BLOCK CONTAINS 0 clause to file descriptions in existing programs could result in a change of behavior in those programs, including some undesirable effects for files opened as INPUT. For this reason, it is recommended that BLOCK0=YES not be set as an installation default.

For further details, see *BLOCK0* in the *Enterprise COBOL Programming Guide*.

BUF

BUF specifies the amount of dynamic storage to be used during compilation.

Syntax



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 1K (1024) bytes.

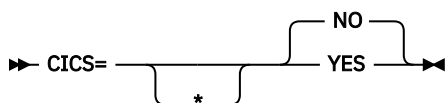
Note:

- BUF cannot exceed the track capacity for the device used, nor can it exceed the maximum allowed by data management services.

CICS

CICS affects whether a COBOL source program that contains CICS statements is to be processed by the integrated CICS translator.

Syntax



Default

CICS=NO

YES

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

NO

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

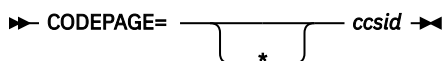
Note:

- The CICS compiler option can contain CICS suboptions. The CICS suboptions delimiters can be quotation marks or apostrophes. CICS suboptions cannot be specified as a COBOL installation default.
- You can specify the CICS compiler option in any of the compiler option sources: installation defaults, compiler invocation, or PROCESS (CBL) statements.
- When the LP(64) compiler option is in effect, the CICS option is not supported. A diagnostic message is emitted if the option is explicitly specified by the user.

CODEPAGE

CODEPAGE affects the coded character set identifier (CCSID) for an EBCDIC code page for processing compile-time and runtime COBOL operations that are sensitive to character encoding.

Syntax



Default

CODEPAGE=1140

ccsid

Specifies a valid coded character set identifier (CCSID) integer that identifies an EBCDIC code page.

The default CCSID 1140 is the equivalent of CCSID 37 (EBCDIC Latin-1, USA), but additionally includes the euro symbol.

Recommendation: To avoid unnecessary conversions and associated performance overhead on systems that use both COBOL and Db2, use the same CODEPAGE compiler option setting as in your Db2 subsystem parameters and application programming defaults (specify in DSNHDECP).

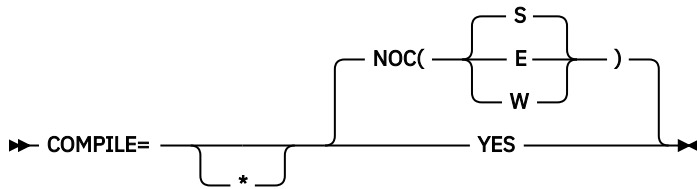
Note: If you specify the TEST option, you must set the CODEPAGE option to the CCSID that is used for the COBOL source program. In particular, programs that use Japanese characters in DBCS literals or DBCS user-defined words must be compiled with the CODEPAGE option set to a Japanese codepage CCSID.

For further details, see *CODEPAGE* in the *Enterprise COBOL Programming Guide*.

COMPILE

COMPILE determines whether compilation continues if diagnostic messages of a specified severity occur.

Syntax



Default

COMPILE=NOC(S)

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.

YES

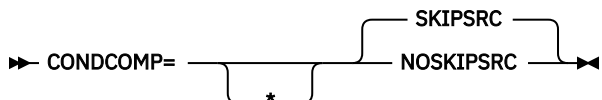
Indicates that you want full compilation, including diagnostics and object code.

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

CONDCOMP

CONDCOMP affects the behavior of conditional compilation directives and controls how conditional code will be displayed in the listing.

Syntax



Default

CONDCOMP=NOSKIPSRC

NOSKIPSRC

If CONDCOMP=NOSKIPSRC is in effect, all source code and comments bounded by conditional compilation directives will be displayed in the listing. Sources lines in false branches of conditional code will be displayed as comments in the listing.

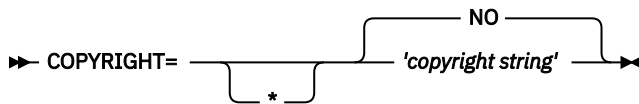
SKIPSRC

If CONDCOMP=SKIPSRC is in effect, some source code and comments bounded by conditional compilation directives will be omitted from the listing. Sources lines and comments in false branches of conditional code will be omitted from the listing.

COPYRIGHT

Use COPYRIGHT to place a string in the object module if the object module is generated. If the object is linked into a program object, the string is loaded into memory with this program object.

Syntax



Default

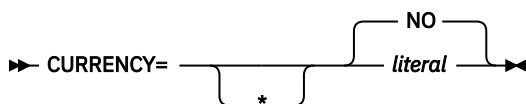
COPYRIGHT=NO

The *copyright string* is limited to 64 characters in length.

CURRENCY

CURRENCY determines whether an alternate currency symbol will be used; the default is dollar sign (\$).

Syntax



Default

CURRENCY=NO

literal

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

The literal must be an alphanumeric literal (optionally a hexadecimal literal) representing a 1-byte EBCDIC character that must not be any of the following items:

- 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 quotation marks or apostrophes regardless of whether the APOST or QUOTE option is in effect.
- 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:

```
.... 0X "''"
```

- The format for a hex literal specification is as follows:

```
X'H1H2' or X"H1H2"
```

where H1H2 is a valid hexadecimal value representing a 1-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.

Note:

- 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.
- 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

DATA affects whether storage for dynamic data areas and other dynamic runtime storage is obtained from above or below the 16 MB line.

Syntax



Default

DATA=31

24

Causes allocation of user data areas in virtual addresses below 16 MB 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:

- A COBOL program is passing items in its WORKING-STORAGE to an AMODE 24 program.
- A 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.

DATA does not affect the location of LOCAL-STORAGE data; the STACK runtime option controls that location instead, along with the AMODE of the program.

31

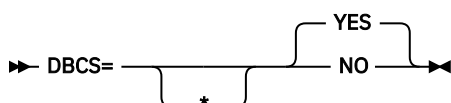
Causes allocation of user data areas, such as WORKING-STORAGE and FD record areas, from unrestricted storage or in space acquired by a GETMAIN with the LOC=ANY option. Specifying this option can result in storage being acquired in virtual addresses either above or below the 16 MB line. The operating system generally satisfies the request with space in virtual addresses above the 16 MB line, if it is available.

Note:

- If a program is compiled with the RENT option, the DATA option controls how space for WORKING-STORAGE and parameter lists is acquired.
- The DATA option has no effect on programs compiled with the NORENT option.
- The DATA option is ignored when LP(64) is in effect. If the user explicitly specifies the DATA option, an informational message is issued.
- The LOCAL-STORAGE section is allocated from stack storage, which is managed by Language Environment. LE allocates stack storage above the 2 GB bar in a 64-bit enclave.

DBCS

DBCS affects whether the compiler recognizes X'0E' and X'0F' in an alphanumeric literal and treats them as shift-out and shift-in control characters for delimiting DBCS data.

Syntax**Default**

DBCS=YES

YES

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

NO

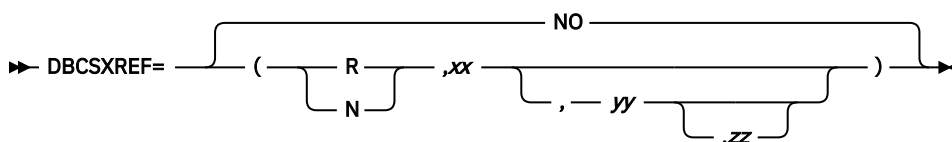
Does not recognize X'0E' and X'0F' as shift-out and shift-in control characters in an alphanumeric literal.

Note:

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

DBCSXREF

DBCSXREF indicates that an ordering program is to be used for cross-referencing of DBCS names.

Syntax**Default**

DBCSXREF=NO

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 program object 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.

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.

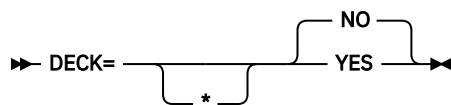
Note:

- The DBCS Ordering Support Program (DBCSOS) must be installed to specify anything other than DBCSXREF=NO.
- If R is specified, ensure that the user region is large enough to accommodate both the compiler and the ordering program.
- Specifying both XREFOPT=NO and DBCSXREF with an ordering program results in a nonzero return code while attempting to assemble the customization macro.
- 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

DECK determines whether 80-column object-code records are produced in a file defined by the SYSPUNCH DD statement.

Syntax



Default

DECK=NO

YES

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

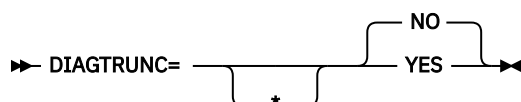
NO

Sends no object code to SYSPUNCH.

DIAGTRUNC

DIAGTRUNC affects whether the compiler issues 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.

Syntax



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

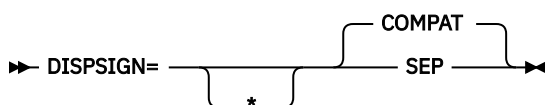
Does not cause the compiler to produce a severity-4 message.

Note:

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

DISPSIGN

The DISPSIGN option controls output formatting for DISPLAY of signed numeric items.

Syntax**Default**

DISPSIGN=COMPAT

COMPAT

If you specify DISPSIGN=COMPAT, formatting for displayed values of signed numeric items is compatible with prior versions of Enterprise COBOL. Overpunch signs are generated in some cases.

SEP

If you specify DISPSIGN=SEP, the displayed values for signed binary, signed packed-decimal, or overpunch signed zoned-decimal items are always formatted with a leading separate sign.

The following example shows the DISPLAY output with the DISPSIGN=COMPAT option or the DISPSIGN=SEP option specified:

<i>Table 3. DISPLAY output with the DISPSIGN=COMPAT option or the DISPSIGN=SEP option specified:</i>		
Data items	DISPLAY output with the DISPSIGN=COMPAT option specified	DISPLAY output with the DISPSIGN=SEP option specified
Unsigned binary	111	111
Positive binary	111	+111
Negative binary	11J	-111
Unsigned packed-decimal	222	222
Positive packed-decimal	222	+222
Negative packed-decimal	22K	-222
Zoned-decimal unsigned	333	333
Zoned-decimal trailing positive	33C	+333
Zoned-decimal trailing negative	33L	-333

Table 3. DISPLAY output with the DISPSIGN=COMPAT option or the DISPSIGN=SEP option specified: (continued)

Data items	DISPLAY output with the DISPSIGN=COMPAT option specified	DISPLAY output with the DISPSIGN=SEP option specified
Zoned-decimal leading positive	C33	+333
Zoned-decimal leading negative	L33	-333

DLL

DLL affects whether an object module generated by the compiler is enabled for dynamic link library (DLL) support.

Syntax



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.

Note: When the LP(64) compiler option is in effect, the following cases occur:

- The DLL option is effectively ignored. Object files generated using LP(64) are DLL enabled. They can be linked as DLL or non-DLL.
- The EXPORTALL option is supported. Use this option to export symbols from programs when building DLLs.

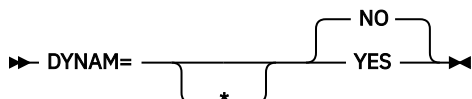
Related references

CALLINTERFACE (*Enterprise COBOL Language Reference*)

DYNAM

DYNAM affects whether the compiler dynamically loads subprograms that are invoked through the CALL *literal* statement.

Syntax



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 program object.

Note:

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

For further details, see DYNAM in the *Enterprise COBOL Programming Guide*.

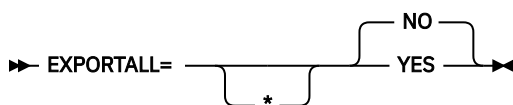
Related references

CALLINTERFACE (*Enterprise COBOL Language Reference*)

EXPORTALL

EXPORTALL affects whether the compiler automatically exports certain symbols when the object deck is link-edited to form a DLL.

Syntax



Default

EXPORTALL=NO

YES

Automatically exports the program-name and alternate entry-point names 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.

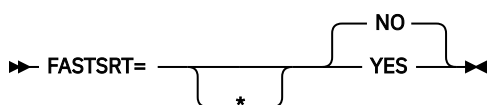
NO

Does not export any symbols.

FASTSRT

FASTSRT determines whether DFSORT or comparable product performs input and output for sort and merge, or whether they are performed by Enterprise COBOL. It applies only to sorting files by using the format 1 SORT statement.

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 *Improving sort performance with FASTSRT* in the *Enterprise COBOL Programming Guide*.)

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

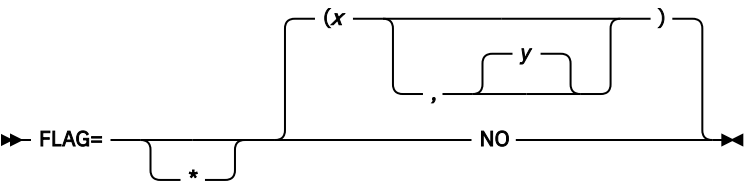
Note:

- 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.
- 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

FLAG affects whether the compiler produces diagnostic messages at or above a specified severity level.

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.

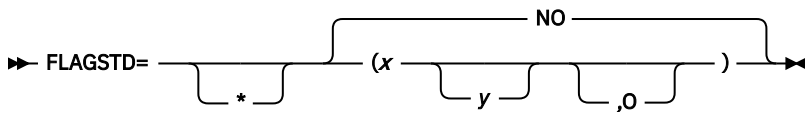
Note:

- 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.
- Specification of FLAG(W|E|S) might result in the loss of entire classes of messages from the Events records in the Associated Data (SYSADATA) file. For further details, see *FLAG* in the *Enterprise COBOL Programming Guide*.

FLAGSTD

FLAGSTD affects the subset of the 85 COBOL Standard language elements that are regarded as conforming, and affects the flagging of informational messages about the language elements used.

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 (where S is a superset of N)

O

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

NO

Specifies that no FIPS flagging is to be done

Note:

- The following elements are flagged as nonconforming and nonstandard IBM extensions to the 85 COBOL Standard:
 - Language syntax used by the COBOL automatic date-processing facilities
 - Language syntax for object orientation and improved interoperability with Java™
 - Use of the PGMNAME=LONGMIXED compiler option

For a complete list of nonconforming and nonstandard elements that is flagged, see *IBM extensions* in the *Enterprise COBOL Language Reference*.

- When FIPS flagging is specified, informational messages in the source program listing identify:

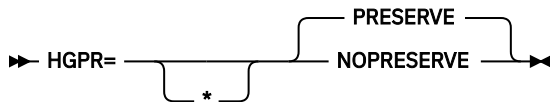
- Whether the language element is obsolete, nonconforming standard, or nonconforming nonstandard (language elements that are both obsolete and nonconforming are flagged as obsolete only)
- The clause, statement, or header containing the nonconforming or obsolete syntax
- The source program line and an indication of the starting column within that line
- The level or optional module to which the language element belongs
- FIPS flagging is suppressed when any error diagnosed as level E or higher occurs.
- Interaction of FLAGSTD and other compiler options:
 - If the following compiler options are explicitly or implicitly specified in a program, FLAGSTD=(other than NO) causes a compiler FIPS message to be issued :
 - ADV=NO
 - BLOCK0=YES
 - CICS=YES
 - DLL=YES
 - DYNAM=NO
 - EXPORTALL=YES
 - FASTSRT=YES
 - LITCHAR=APOST
 - NAME=NO
 - NUMPROC=PFD
 - PGMNAME=LONGMIXED
 - QUALIFY=EXTEND
 - THREAD=YES
 - TRUNC=OPT or BIN
 - VLR=COMPAT
 - WORD=(other than NO or RWT)
 - INVDATA=(other than NO)
 - 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
 - DBCS=YES
 - DYNAM=NO
 - LITCHAR=APOST
 - NUM=YES
 - NUMPROC=PFD
 - QUALIFY=EXTEND
 - SEQ=YES
 - TRUNC=OPT or BIN
 - VLR=COMPAT
 - WORD=(other than NO or RWT)
 - INVDATA=(other than NO)
 - ZWB=NO
- FLAGSTD might produce events records in the Associated Data file for FIPS standard conformation messages. Error messages are not guaranteed to be sequential with respect to source record numbers.

FLAGSTD messages can be converted into diagnostic messages, or suppressed. For details, see [“MSGEXIT” on page 44](#).

HGPR

The HGPR option controls the compiler usage of the 64-bit registers provided by IBM z/Architecture processors.

Syntax



Default is: HGPR=PRESERVE

The Enterprise COBOL compiler uses the 64-bit width of the IBM z/Architecture General Purpose Registers (GPRs). HGPR stands for "High-halves of 64-bit GPRs", which means the use of native 64-bit instructions.

HGPR=PRESERVE

If you specify `HGPR=PRESERVE`, the compiler preserves the high halves of the 64-bit GPRs that a program is using, by saving them in the prolog for the function and restoring them in the epilog. The `PRESERVE` suboption is necessary only if the caller of the program is not Enterprise COBOL, Enterprise PL/I, or z/OS XL C/C++ compiler-generated code.

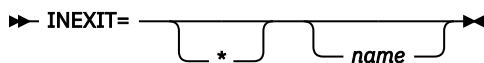
HGPR=NOPRESERVE

If you specify `HGPR=NOPRESERVE`, the compiler omits preserving the high-halves of the 64-bit GPRs that a program is using, which improves performance.

INEXIT

INEXIT designates a module to be called to obtain source statements instead of reading the SYSIN data set.

Syntax



Default

No exit is specified. Equivalent to specifying the NOINEXIT suboption of the EXIT compiler option. If INEXIT=* is coded without the *name* parameter, NOINEXIT cannot be overridden.

name

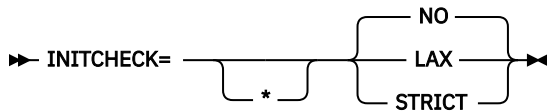
Identifies a module to be used with the EXIT compiler option. If the suboption for this user exit is specified, the compiler loads the named module and calls it to obtain source statements instead of reading the SYSIN data set. If the option is supplied, the SYSIN data set is not opened.

For more information about the EXIT compiler option, see *EXIT compiler option* in the *Enterprise COBOL Programming Guide*.

INITCHECK

Use the INITCHECK option to have the compiler check for uninitialized data items and issue warning messages when they are used without being initialized.

Syntax



Default

INITCHECK=NO

NO

The compiler will not issue any warning messages for uninitialized data items.

LAX

The compiler will check for uninitialized data items and issue a warning message when a data item is used without being initialized. However, if a data item is initialized on at least one logical path to a statement, no warning message will be issued.

STRICT

The compiler will still check for uninitialized data items and issue a warning message when a data item is used without being initialized. However, unlike INITCHECK=LAX, INITCHECK=STRICT will issue a warning message about uninitialized data for a data item used in a statement unless the data item is initialized on all logical paths to the statement.

Here is a sample program to illustrate the behavior differences between specifying INITCHECK=LAX versus INITCHECK=STRICT. Y and Z represent some data items, with no value clauses:

```
PROCEDURE DIVISION.  
  IF Y > 5  
    MOVE 2 TO Z  
  END-IF  
  DISPLAY Z
```

Z is initialized on one path to the DISPLAY statement but not the other, so if INITCHECK=LAX is in effect, a warning message will be issued for Y only, while INITCHECK=STRICT will also issue a warning message for Z.

Restrictions:

- The INITCHECK option analyzes data items in the WORKING-STORAGE SECTION and LOCAL-STORAGE SECTION only. In particular, it does not analyze data items in the LINKAGE SECTION or FILE SECTION.
- The INITCHECK analysis does not track external or global data items.
- The INITCHECK analysis does not track individual elements in tables independently. Instead, if one element of a table is initialized, all corresponding elements of the table are considered to be initialized. This applies to both fixed-length and variable-length tables.
- The INITCHECK analysis does not track the initialization of items if it happens through a pointer. For example, if a pointer to an uninitialized data item is created by using ADDRESS-OF, and that data item is initialized through that pointer, the INITCHECK analysis might also issue a warning message.
- For uninitialized data items being passed BY REFERENCE, no warning messages will be issued. However, the INITCHECK analysis will warn about uninitialized data items being passed BY CONTENT and BY VALUE.
- If a data item is in a group with other items that have had their address taken, for example, as the result of being an SQL host variable, then that data item will also be considered to have its address taken, and the set of all address taken data items is always considered to be set by any call to an external function.

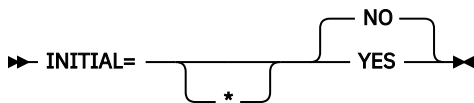
Notes:

- All of the INITCHECK analyses occur at compile time only.
- The INITCHECK option has no effect on the behavior or performance of the program after it has been compiled.
- Use of the INITCHECK option might increase compile time and memory consumption.
- The INITCHECK option reports and prints only the first uninitialized data item in a group. Subsequent data items that are also uninitialized will not be printed.
- INITCHECK is more accurate when used with OPT=1 or OPT=2, but it is also helpful when used with OPT=0.

INITIAL

The INITIAL compiler option causes a program and all of its nested programs to behave as if the IS INITIAL clause was specified on the PROGRAM-ID paragraph.

Syntax



Default is: INITIAL=NO

YES

INITIAL=YES causes a program and all of its nested programs to behave as if the IS INITIAL clause was specified on the PROGRAM-ID paragraph.

Note: INITIAL=YES and the IS INITIAL clause have no effect on data items that do not have VALUE clauses.

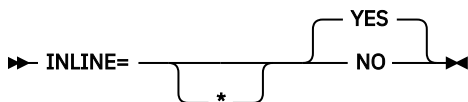
NO

INITIAL=NO will have no effect on programs that already have IS INITIAL on the PROGRAM-ID paragraph in the source.

INLINE

The INLINE option controls whether the inlining of procedures (paragraphs or sections) referenced by PERFORM statements in the source program is allowed.

Syntax



Default is: INLINE=YES

YES

If you specify INLINE=YES, when OPTIMIZE (1) or OPTIMIZE (2) is in effect, the compiler can inline procedures referenced by PERFORM statements in the source program. INLINE is a potential performance-boosting option. Note that INLINE was always in effect in COBOL 5.

NO

If you specify INLINE=NO, the compiler is prevented from *inlining*¹ procedures referenced by PERFORM statements in the source program, regardless of the optimization level in effect. INLINE=NO should only be set as the default if a particular reason has been identified to eliminate all inlining of PERFORM statements in programs. This is rare and not recommended.

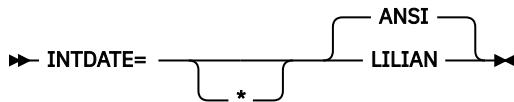
Note:

1. The word *inlining* here implies that the compiler might choose to replace the PERFORM of a procedure (paragraph or section) with a copy of that procedure's code. By inserting the procedure code at the location of the PERFORM, the compiler saves the overhead of branching logic to and from the procedure.

INTDATE

INTDATE affects the starting date that is used for date intrinsic functions.

Syntax



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 1.1.

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 1.1.

Note:

- 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.
- If you set your installation option to INTDATE(LILIAN), you should recompile all of your COBOL/370 1.1 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.

INVDATA

The INVDATA option tells the compiler whether the data in USAGE DISPLAY and PACKED-DECIMAL data items is valid, and if not, what the behavior of the compiler should be.

Because most users have valid data in their USAGE DISPLAY and USAGE PACKED-DECIMAL data items, they should use NOINVDATA, even if they use NUMPROC=NOPFD. Even if you find that your programs are processing invalid data at run time with the NUMCHECK compiler option, you should change your programs to avoid processing invalid data and use NOINVDATA.

Note: The goal of the INVDATA option is to provide a behavior that is as compatible as possible with the behavior of programs compiled with COBOL 4 or earlier versions in cases of invalid numeric data. When discrepancies are found, this option will be updated in favor of making the behavior more closely match the behavior of COBOL 4 or earlier versions.

When the INVDATA option is in effect, the compiler will avoid performing known optimizations that might produce a different result than COBOL 4 or earlier versions when a zoned decimal or packed decimal data item has invalid digits or an invalid sign code, or when a zoned decimal data item has invalid zone bits.

The following table provides a quick reference on how to set the INVDATA and NUMPROC options when migrating to COBOL 6.2 or later versions from earlier versions of COBOL, depending on the default value

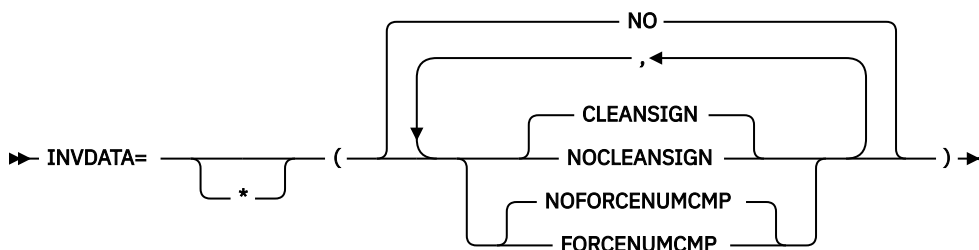
of the NUMPROC option that was used in the earlier version of COBOL and whether or not you have invalid data.

Table 4. Setting INVDATA and NUMPROC options when migrating from earlier COBOL versions

COBOL versions	Invalid data present?	NUMPROC/ZONEDATA used in COBOL 6.1 or earlier versions	INVDATA and NUMPROC settings in COBOL 6.2 or later versions
Pre-COBOL 5	No	NUMPROC=MIG	INVDATA=(NO),NUMPROC=NOPFD
Pre-COBOL 5	No	NUMPROC=NOPFD	INVDATA=(NO),NUMPROC=NOPFD
Pre-COBOL 5	No	NUMPROC=PFD	INVDATA=(NO),NUMPROC=PFD
Pre-COBOL 5	Yes	NUMPROC=MIG	INVDATA=(FORCENUMCMP, NOCLEANSIGN), NUMPROC=NOPFD
Pre-COBOL 5	Yes	NUMPROC=NOPFD	INVDATA=(NOFORCENUMCMP, CLEAN SIGN), NUMPROC=NOPFD
Pre-COBOL 5	Yes	NUMPROC=PFD	INVDATA=(NOFORCENUMCMP, CLEAN SIGN), NUMPROC=PFD
COBOL 5 or later	No	ZONEDATA=PFD	INVDATA=(NO)
COBOL 5 or later	Yes	ZONEDATA=NOPFD	INVDATA=(NOFORCENUMCMP, CLEAN SIGN)
COBOL 5 or later	Yes	ZONEDATA=MIG	INVDATA=(FORCENUMCMP, CLEAN SIGN) ¹

1. INVDATA=(FORCENUMCMP, NOCLEANSIGN) is a closer representation of the pre-COBOL 5 NUMPROC=MIG behavior than INVDATA=(FORCENUMCMP, CLEAN SIGN) when invalid data is present. If you are not satisfied with the behavior of ZONEDATA=MIG in COBOL 5 or later versions when invalid data is present, then consider using INVDATA=(FORCENUMCMP, NOCLEANSIGN) to more closely mimic the pre-COBOL 5 NUMPROC=MIG behavior when invalid data is present.

Syntax



Default

INVDATA=(NO)

Specify at least one suboption to enable INVDATA.

Suboption defaults are: NOFORCENUMCMP and CLEAN SIGN.

(NO)

When INVDATA=(NO) is in effect, the compiler assumes that all data in USAGE DISPLAY and PACKED-DECIMAL data items is valid, and generates the most efficient code possible. For example, the compiler might generate a string comparison to avoid numeric conversion.

(FORCENUMCMP | NOFORCENUMCMP)

When INVDATA=(FORCENUMCMP) is in effect, the compiler generates instructions to do comparisons of zoned decimal data items that ignore the zone bits of each digit. For example, the zoned decimal value is converted to packed-decimal with a PACK instruction before the comparison.

When INVDATA=(NOFORCENUMCMP) is in effect, the compiler generates instructions for numeric comparisons or an alphanumeric comparison of zoned-decimal data in the same manner as COBOL 4 or earlier versions do when using NUMPROC=NOPFD | PFD with COBOL 4 or earlier versions:

- In the cases where COBOL 4 or earlier versions considered the zone bits, the compiler generates an alphanumeric comparison which will also consider the zone bits of each digit in zoned decimal data items. The zoned decimal value remains as zoned decimal.
- In the cases where COBOL 4 or earlier versions ignored the zone bits, the compiler generates numeric comparisons that ignore the zone bits of each digit in zoned decimal data items. The zoned-decimal value is converted to packed decimal with a PACK instruction before the comparison.

In order for the compiler to handle zone bits in the same way as COBOL 4 or earlier versions did when generating comparisons of zoned-decimal data, the NUMPROC suboption used in COBOL 6 must match the NUMPROC suboption used in COBOL 4 or earlier versions:

- To get the COBOL 4 or earlier versions NUMPROC=NOPFD behavior in COBOL 6, use INVDATA=(NOFORCENUMCMP) and NUMPROC=(NOPFD) in COBOL 6.
- To get the COBOL 4 or earlier versions NUMPROC=PFD behavior in COBOL 6, use INVDATA=(NOFORCENUMCMP) and NUMPROC=PFD in COBOL 6.

Note: The sign code must be a valid sign code according to the NUMPROC compiler option setting. In addition, the low-order byte must have a valid zone (x'F') for unsigned and signed with either SIGN IS LEADING or SIGN IS SEPARATE.

(CLEANSIGN | NOCLEANSIGN)

When the INVDATA=(CLEANSIGN) option is in effect, the compiler generates code to clean the sign nibble of USAGE DISPLAY and USAGE PACKED-DECIMAL data items on input to compare, add, subtract, multiply, and divide operations.

Note: CLEANSIGN does not apply to USAGE DISPLAY items defined with SIGN IS SEPARATE.

When the INVDATA=(NOCLEANSIGN) option is in effect, the compiler avoids generating code to clean the sign nibble of USAGE DISPLAY and USAGE PACKED-DECIMAL data items on input to compare, add, subtract, multiply, and divide operations, increasing the probability of a SOC7 abend when one of the operands of the operation contains an invalid sign nibble.

Note: The INVDATA option affects the behavior of MOVE statements, comparisons, and computations for USAGE DISPLAY or PACKED-DECIMAL data items that could contain invalid digits, an invalid sign code, or invalid zone bits.

In the following example, you can see a data item with an invalid zone bit 4 in the zone bits in the middle of data item VALUE1, forced in by REDEFINES:

```
77 VALUE0    PIC X(4) VALUE '00 0'.          *>  x'F0F040F0'
77 VALUE1    REDEFINES VALUE0 PIC 9(4).
PROCEDURE DIVISION.
  IF VALUE1 = ZERO
    DISPLAY 'INVDATA(FORCENUMCMP) is in effect ' VALUE1
  ELSE
    DISPLAY 'INVDATA(NOFORCENUMCMP) is in effect ' VALUE1
  END-IFCopy code
```

In this example,

- With COBOL 4 or earlier versions, the test is true if the NUMPROC=MIG option is used, and false for NUMPROC=NOPFD | PFD.
- With COBOL 5 or later versions:
 - When using INVDATA=(NO), the test is true at OPT=(0) and false at OPT=(1 | 2).
 - When using INVDATA=(NOFORCENUMCMP), the test is false at any OPT setting.

In all, to ease your migration to COBOL 6:

- If your digits, sign codes, and zone bits are valid, use INVDATA=(NO), and if you used NUMPROC=PFD or NUMPROC=NOPFD in COBOL 4 or earlier versions, then use the same NUMPROC setting when using COBOL 6; if you used NUMPROC=(MIG) in COBOL 4 or earlier versions, then use NUMPROC=NOPFD when using COBOL 6.
- If you have invalid digits, invalid sign codes, or invalid zone bits in your data, change your programs or systems so that your programs do not have invalid data in numeric data items at run time.

When you have corrected your programs or systems, you can use the preferred INVDATA=(NO) option. Only if you cannot contain this work and must continue to run with invalid data, consider the following choices for INVDATA:

- If you used NUMPROC=MIG with COBOL 4 or earlier versions, use INVDATA=(FORCENUMCMP,NOCLEANSIGN) and NUMPROC=NOPFD with COBOL 6.
- If you used NUMPROC=NOPFD with COBOL 4 or earlier versions, use INVDATA=(NOFORCENUMCMP,CLEANSIGN) and NUMPROC=NOPFD with COBOL 6.
- If you used NUMPROC=PFD with COBOL 4 or earlier versions, use INVDATA=(NOFORCENUMCMP,CLEANSIGN) and NUMPROC=PFD with COBOL 6.

Notes:

- If you completed migration from COBOL 4 or earlier versions to COBOL 5 or 6 in the past and used the deprecated ZONEDATA=MIG option in COBOL 5 or 6 and are satisfied with the behavior, use INVDATA=(FORCENUMCMP,CLEANSIGN) now instead of ZONEDATA=MIG.
- It is not always possible to entirely match the behavior of the old compiler even with these options when faced with clearly invalid data. For example, even for comparisons, INVDATA=(NOFORCENUMCMP) does not give the same result in all cases as COBOL 4 does.

Performance consideration: INVDATA=(NO) gives better runtime performance than INVDATA=(NOFORCENUMCMP | FORCENUMCMP,NOCLEANSIGN | CLEAN SIGN) does. INVDATA=(NOFORCENUMCMP | FORCENUMCMP,NOCLEANSIGN | CLEAN SIGN) disables some of the optimizations that NUMPROC=PFD can give.

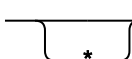
related references

[“NUMPROC” on page 50](#)

LANGUAGE

LANGUAGE affects the language used for compiler output messages.

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 5. 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:

- The LANGUAGE option name must consist of at least the first two identifying characters. Other characters after the first two can be used; however, only the first two are used to determine the language name.
- This compiler option does not affect the language in which runtime messages are displayed. For more information about runtime options and messages, see the *z/OS Language Environment Programming Guide*.
- Some printers use only uppercase and might not accept output in mixed case (LANGUAGE=ENGLISH).
- To specify the Japanese language option, the Japanese National Language Feature must be installed.
- To specify the English language option (mixed-case English), the U.S. English Language Feature must be installed.
- 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.
- The selection of Japanese together with specification of the ADATA option might result in DBCS characters being written to error identification records in the Associated Data file.
- To change to uppercase English or Japanese compiler messages, in addition to using the LANGUAGE compiler option, you must also set the Language Environment runtime option NATLANG at compile time. We recommend using CEEOPTS DD in the compile JCL.

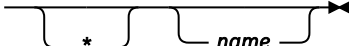
For example, to change messages to Japanese, use LANGUAGE=JA and also specify the NATLANG LE runtime option at compile time:

```
//CEEOPTS DD *
           NATLANG(JPN)
/*
```

LIBEXIT

LIBEXIT designates a module to be called to obtain COPY statements instead of reading the SYSLIB or library-name data set.

Syntax

➔ LIBEXIT= 

Default

No exit is specified. Equivalent to specifying the NOLIBEXIT suboption of the EXIT compiler option. If LIBEXIT=* is coded without the *name* parameter, NOLIBEXIT cannot be overridden.

name

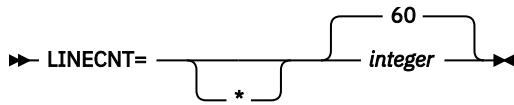
Identifies a module to be used with the EXIT compiler option. If the suboption for this user exit is specified, the compiler loads the named module and calls it to obtain COPY statements instead of reading the SYSLIB or library-name data set. If the option is supplied, the SYSLIB and library-name data sets are not opened.

For more information about the EXIT compiler option, see *EXIT compiler option* in the *Enterprise COBOL Programming Guide*.

LINECNT

LINECNT affects the number of lines to be printed on each page of the compiler source listing.

Syntax



Default

LINECNT=60

integer

Specifies the number of lines to be printed on each page of the compiler source code listing, and it must be an integer between 10 and 255, or 0. 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.

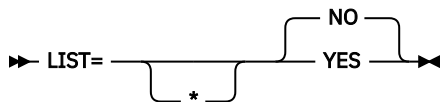
If you specify LINECNT=0, no page ejects are generated in the compilation listing.

The LINECNT installation option is equivalent to the LINECOUNT compiler option.

LIST

LIST affects whether an assembler-language expansion is produced in source listings.

Syntax



Default

LIST=NO

YES

Produces a listing that includes:

- The assembler-language expansion of source code
- Constant area
- Program prolog areas (PPA1, PPA2, PPA3, PPA4)
- Time stamp, compiler version, and build level information
- Compiler options and program information
- Base locator table
- External symbols dictionary
- Initial heap storage maps
- Stack storage maps

NO

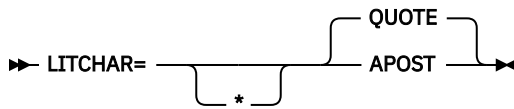
Suppresses this listing.

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

LITCHAR affects whether the QUOTE figurative constant represents quotation marks or apostrophes.

Syntax



Default

LITCHAR=QUOTE

APOST

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

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 85 COBOL Standard.

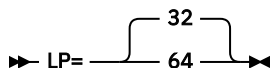
Note:

- Either quotation marks or apostrophes can be used as literal delimiters, regardless of whether the APOST or QUOTE option is in effect.
- The delimiter character used as the opening delimiter for a literal must be used as the closing delimiter for that literal.

LP

The LP compiler option affects whether a AMODE 31 (31-bit) or AMODE 64 (64-bit) program should be generated with the related language features enabled when compiling COBOL programs.

Syntax



You can specify the LP option in the ways that you specify other compiler options. However, if you specify the option in a CBL (PROCESS) statement, you can only specify the LP option for the first program. You cannot change the value of the option for subsequent programs in the batch.

Default

LP=32

32

Indicates that an AMODE 31 (31-bit) program should be generated with the related language features enabled.

64

Indicates that an AMODE 64 (64-bit) program should be generated with the related language features enabled.

Runtime consideration: Currently, Language Environment does not support mixing AMODE 64 and AMODE 31 programs in the same application. If one program is compiled with LP(64), all programs within the application should also be compiled with LP(64). For static CALLs, the binder will issue a message if it encounters mixing addressing modes during external name resolution. For dynamic CALLs, you would receive a run time error for a CALL between programs if one is AMODE 64 and one is AMODE 31 or 24.

When using the LP=64 compiler option, the compilation process includes a component that runs in POSIX(ON) mode. This implies that there must be an OMVS Segment established in RACF® (or equivalent in RACF alternatives) for each user executing the compiler with this option.

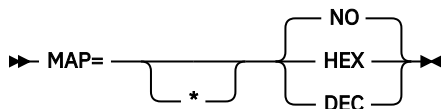
Restrictions for programs using LP=64: Programs compiled using the LP=64 compiler option cannot contain XML GENERATE or XML PARSE statements, JSON GENERATE or JSON PARSE statements, Object-oriented COBOL statements, ALTER statements, GO TO. statements, and DISPLAY ... UPON SYSPUNCH statements. In addition, AMODE 64 programs cannot run in CICS or IMS.

Note: Under LP(64), some compiler options are not applicable. For more information, see Using compiler options to compile AMODE 64 programs in the *Enterprise COBOL Programming Guide*.

MAP

The MAP option affects whether map information about the DATA DIVISION items and all implicitly declared items is shown in the listing. The option also controls whether hexadecimal or decimal offsets are shown for MAP output in the listing.

Syntax



Default

MAP=NO

HEX or DEC

Maps items that are declared in the DATA DIVISION. Map output includes:

- DATA DIVISION map
- Nested program structure map, and program attributes
- Size of the program's WORKING-STORAGE and LOCAL-STORAGE and its location in the object code if the program is compiled with the NORENT option

If you specify MAP=HEX, data item offsets within groups will be in hexadecimal notation.

If you specify MAP=DEC, data item offsets within groups will be in decimal notation.

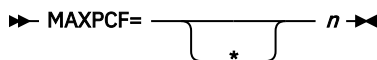
NO

Mapping is not performed.

MAXPCF

Use the MAXPCF option to specify a maximum program complexity factor value. The program complexity factor (PCF) is computed by the compiler and the computed value is in the listing file. If the PCF of your program exceeds the maximum value, the compiler will automatically reduce the optimization level to speed up the compilation and use less storage. Therefore, when you compile a suite of programs, you do not have to specify an OPTIMIZE option value for each program.

Syntax



Default

MAXPCF=100000

n

Must be an integer of 0 - 999999.

The aspects of the program taken into consideration when computing the complexity factor include:

- The number of COBOL statements in the PROCEDURE DIVISION, including generated statements from the CICS, SQL or SQLIMS options, and the expansion of COPY and REPLACE statements
- Initialization operations for WORKING-STORAGE or LOCAL-STORAGE data items with value clauses

- Operations for variable-length groups or subgroups in the DATA DIVISION, which compute their size at run time

Note: PCF is not a metric to measure how complex a program is. It is merely a count of COBOL items that can cause problems for optimization when there are a lot of them. To measure program complexity, you should use something like the [Metrics](#) feature provided by IBM Developer for z/OS.

For large and complex programs, you can use the MAXPCF option to set a threshold on the program complexity that the compiler attempts to optimize. Lower the MAXPCF value to reduce the optimization level, hence the compiler needs less memory and compilation time. Raise the MAXPCF value to attempt to optimize the programs at the cost of longer compilation time.

If you specify MAXPCF=0, no limit is enforced on the complexity of the program, and the MAXPCF option has no effect.

If you specify MAXPCF=*n* and *n* is not zero, when the program complexity factor exceeds *n*, any specification of OPTIMIZE (1) or OPTIMIZE (2) is reset to OPTIMIZE (0), and a warning message is generated.

If the COBOL source file contains a sequence of source programs (a batch compile), the MAXPCF limit is applied on a per program basis.

Notes:

- If the OPT=1 or OPT=2 option is set at installation time as a fixed, nonoverridable option, then MAXPCF=*n* with a nonzero *n* is an option conflict. In this case, the OPTIMIZE option takes precedence and the MAXPCF=0 option is forced on.
- If you attempt to optimize a program larger than the default threshold by raising the value of MAXPCF to *n* where *n* is greater than the default, or by specifying MAXPCF (0), the compiler might take excessive time to compile or fail to compile because of insufficient memory.

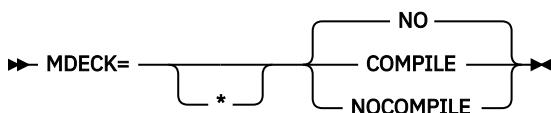
Related references

[“OPTIMIZE” on page 51](#)

MDECK

The MDECK compiler option specifies that a copy of the updated input source after library processing (that is, the result of COPY, BASIS, REPLACE, EXEC SQL INCLUDE, EXEC SQLIMS INCLUDE, and conditional compilation directive statements) is written to a file.

Syntax



Default

MDECK=NO

COMPILE

Compilation continues normally after library processing and generation of the MDECK output file.

NOCOMPILE

Compilation ends after library processing is completed and the expanded source program file is written.


NO

An MDECK output file is not produced.

MSGEXIT

MSGEXIT designates a module to be called to enable customization of compiler messages.

Syntax

➤ MSGEXIT= 

Default

No exit is specified. Equivalent to specifying the NOMSGEXIT suboption of the EXIT compiler option. If MSGEXIT=* is coded without the *name* parameter, NOMSGEXIT cannot be overridden.

name

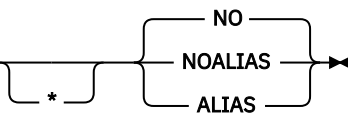
Identifies a module to be used with the EXIT compiler option. If the suboption for this user exit is specified, the compiler loads the named module and calls it to enable customization of compiler messages. The severity of messages can be changed, messages can be suppressed, and FIPS messages resulting from the FLAGSTD compiler option can be converted into diagnostic messages.

For more information about the EXIT compiler option, see *EXIT compiler option* in the *Enterprise COBOL Programming Guide*.

NAME

NAME affects whether a program management binder NAME statement is appended to each object module and whether an ALIAS statement is created for each ENTRY statement.

Syntax

➤ NAME= 

Default

NAME=NO

ALIAS

Creates a program management binder ALIAS statement for each ENTRY statement in the program. The ALIAS statement is inserted preceding the NAME statement corresponding to the PROGRAM-ID.

NOALIAS

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

NO

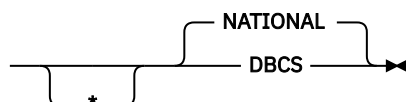
Does not append program management binder NAME statements.

The NAME option lets you create multiple modules in a program library with a single batch compilation, which can be useful for dynamic calls.

NSYMBOL

NSYMBOL controls the interpretation of the N symbols used in PICTURE clauses, indicating whether national or DBCS processing is assumed.

Syntax

➤ NSYMBOL= 

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

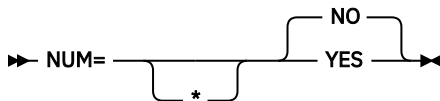
Use NATIONAL when data items are defined with the PICTURE clause consisting only of the PICTURE symbol N and without the USAGE clause. Such data items are treated as if the USAGE NATIONAL clause were specified. Literals of the form N". ." or N'. .' are treated as national literals.

Note:

- The NSYMBOL(DBCS) option is compatible with previous releases of IBM COBOL. The NSYMBOL(NATIONAL) option handles the N symbol consistently with the 2002 COBOL Standard.
- NSYMBOL(NATIONAL) forces the DBCS option.

NUM

NUM affects whether source-program line numbers are used in error messages and procedure maps.

Syntax**Default**

NUM=NO

YES

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

NO

Uses the compiler-generated line numbers in error messages and procedure maps.

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.

NUMCHECK

The NUMCHECK compiler option tells the compiler whether to generate extra code to validate data items when they are used as sending data items. For zoned decimal (numeric USAGE DISPLAY) and packed decimal (COMP-3) data items, the compiler generates implicit numeric class tests for each sending field. For alphanumeric senders whose contents are being moved to a numeric receiver, the compiler treats the sender as a numeric integer so NUMCHECK generates an implicit numeric class test for each alphanumeric sender. For binary data items, the compiler generates SIZE ERROR checking to see whether the data item has more digits than its PICTURE clause allows.

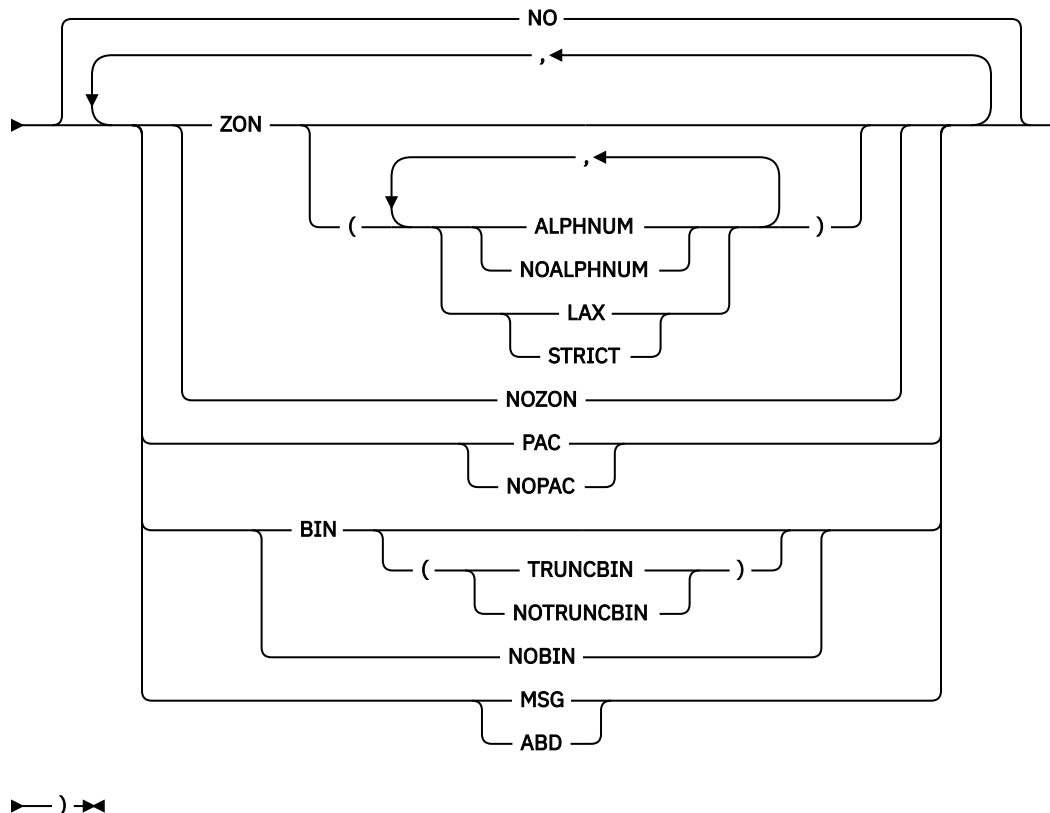
The NUMCHECK option is updated to remove redundant checks for invalid data, thus improving runtime performance. There may be fewer runtime messages than before.

The analysis done to remove redundant checks is more involved at OPT(1|2) than at OPT(0). OPT(0) does a simpler form of the analysis to keep compilation time as low as possible. There may be fewer messages at higher OPT levels.

When the compiler is able to determine at compile time that a check will always find invalid data, a compile-time message is produced and the runtime check is removed. (See MSG | ABD below.)

Syntax

➡ NUMCHECK= (→



Default

NUMCHECK=(NO)

Suboption defaults are:

- If no suboption is specified, defaults are
NUMCHECK=(ZON(ALPHNUM, STRICT), PAC, BIN(TRUNCBIN), MSG).
- If no datatype suboption is specified, default datatype suboptions are ZON(ALPHNUM, STRICT), PAC, and BIN(TRUNCBIN). For example, NUMCHECK=(ABD) has the same effect as NUMCHECK=(ZON(ALPHNUM, STRICT), PAC, BIN(TRUNCBIN), ABD).
- If only one datatype suboption is specified, defaults are NOZON, NOPAC, NOBIN, and MSG. For example, NUMCHECK=(BIN) has the same effect as NUMCHECK=(NOZON, NOPAC, BIN(TRUNCBIN), MSG).
- If all datatype suboptions are specified with NO, then the listing will show NONUMCHECK. For example, NUMCHECK=(NOZON, NOPAC, NOBIN) has the same effect as NUMCHECK=NO.

ZON(ALPHNUM|NOALPHNUM, LAX|STRICT) | NOZON

Specifying ZON(ALPHNUM) causes the compiler to generate code for an implicit numeric class test for zoned decimal (numeric USAGE DISPLAY) data items that are used as sending data items in COBOL statements.

Specifying ZON(NOALPHNUM) causes the compiler to generate code for an implicit numeric class test for zoned decimal (numeric USAGE DISPLAY) data items that are used as sending data items in COBOL statements, except when they are used in a comparison with an alphanumeric data item, alphanumeric literal or alphanumeric figurative constant.

Receivers are not checked, unless they are both a sender and a receiver, such as data item B in the following sample statements:

```
ADD A TO B
```

```
DIVIDE A INTO B
```

```
COMPUTE B = A + B
```

```
INITIALIZE B REPLACING ALPHANUMERIC BY B
```

This checking is done before the data is used in each statement:

- If the data is NOT NUMERIC, either a warning message for NUMCHECK=(ZON,MSG) or a terminating message for NUMCHECK=(ZON,ABD) is issued.
- If the data is NUMERIC, the external behavior of the statement is the same as NUMCHECK=(NOZON), other than being slower.

Specifying ZON(LAX) causes the compiler to be more tolerant of invalid data in a zoned decimal data item. Three cases are considered by the compiler as follows:

- An unsigned zoned decimal data item redefines a signed trailing overpunch zoned decimal data item such that the last byte of the unsigned item overlaps the last byte of the signed item. In this case, the unsigned redefining item is treated as a signed zoned decimal item for the purposes of the NUMCHECK checking.

Notes:

- The signed zoned decimal item that is redefined must be a level-01 or level-77 item. The unsigned zoned decimal item can be a level-01 or level-77 item or can be a subordinate item in a group.
- The unsigned zoned decimal item does not need to overlap the entire signed zoned decimal item. It is only necessary for the last byte of each item to overlap. For example:

```
01 NUM1 PIC S9(8).  
01 NUM2 REDEFINES NUM1.  
    03 NUM2-PART1 PIC 9(4).  
    03 NUM2-PART2 PIC 9(2).  
    03 NUM2-PART3 PIC 9(2).
```

In this case, data item NUM2-PART3 will be treated by NUMCHECK as a signed zoned decimal data item because its last byte overlaps the last byte of NUM1, which is a signed trailing overpunch zoned decimal item. Thus, the following values of NUM2-PART3 are all considered valid:

- x'F1F2F3F4F5F6F7F8'
- x'F1F2F3F4F5F6F7C8'
- x'F1F2F3F4F5F6F7D8'

- A zoned decimal data item redefines a numeric-edited data item that may contain leading spaces, as indicated by the Z symbol in the numeric-edited item's PICTURE string, and the leading bytes of the zoned decimal data item overlap some or all of the leading bytes of the numeric-edited item. In this case, NUMCHECK will tolerate spaces in the leading bytes of the zoned decimal data item that overlap those bytes of the numeric-edited item that permit spaces.

Notes:

- The numeric-edited item that is redefined must be a level-01 or level-77 item. The zoned decimal item can be a level-01 or level-77 item or can be a subordinate item in a group.
- If the zoned decimal item is signed, it must be signed trailing overpunch.
- The first byte of the zoned decimal item must overlap the first byte of the numeric-edited item to be considered eligible for this treatment, but the zoned decimal item does not need to overlap the entire numeric-edited item. For example:

```

01 NUMED PIC ZZ99.99.
01 NUM REDEFINES NUMED.
   03 INTVAL PIC 9(4).
   03 FILLER PIC X.
   03 DECVAL PIC 9(2).

```

In this case, NUMCHECK tolerates spaces in the first two bytes of INTVAL because it overlaps the first two bytes of NUMED which are defined with the Z symbol in its PICTURE string. Thus, the following values of INTVAL are all considered valid:

- x'F1F2F3F4'
- x'40F1F2F3'
- x'4040F1F2'

Note that for performance reasons, mixes of spaces and non-spaces are tolerated in the leading bytes, thus x'F140F1F2' is also considered valid.

- A zoned decimal data item is moved to another zoned decimal data item. In this case, NUMCHECK will not check the sender of the move. However, if the sender is subsequently used in a numeric context, it will be checked.

If ZON(STRICT) is specified, NUMCHECK does not consider any data items that a zoned decimal data item might redefine, and strict checking of the zoned decimal data is performed as usual.

Note: The ZON(LAXREDEF|STRICTREDEF) option is deprecated but is tolerated for compatibility, and it is replaced by the ZON(LAX|STRICT) option.

PAC | NOPAC

Specifying PAC causes the compiler to generate code for an implicit numeric class test for packed decimal (COMP-3) data items that are used as sending data items in COBOL statements. For packed decimal data items that have an even number of digits, the unused bits are checked for ones.

Restriction: For CALL statements, NUMCHECK=(ZON) and NUMCHECK=(PAC) check BY CONTENT data items that are zoned decimal or packed decimal, but they do not check BY REFERENCE parameters. (Neither zoned decimal nor packed decimal data items can be specified in a BY VALUE phrase.)

BIN(TRUNCBIN|NOTRUNCBIN) | NOBIN

Specifying BIN causes the compiler to generate code similar to ON SIZE ERROR to test if binary data items contents are bigger than the PICTURE clause. This extra code will be generated only for binary data items that are used as sending data items, and COMP-5 data items will not get this ON SIZE ERROR code generated.

When BIN(TRUNCBIN) is in effect, the checking code is generated for binary data items, even when the TRUNC=(BIN) compiler option is in effect. Note that BIN(TRUNCBIN) is the default when no suboption for BIN is specified.

When BIN(NOTRUNCBIN) is in effect, the checking code is not generated for binary items when the TRUNC=(BIN) compiler option is in effect.

Note: BIN(NOTRUNCBIN) is useful for users who want to make NUMCHECK=(... , BIN, ...) a fixed option in their default options, but do not want the checking to be done for modules that are compiled with the TRUNC=(BIN) option in effect.

MSG | ABD

Determines whether the message issued for invalid data is a warning level message to continue processing or a terminating level message to cause an abend:

- If MSG is in effect, a runtime warning message with the line number, data item name, data item content, and program name is issued. Also, the affected statements will still be executed.
- If ABD is in effect, a terminating message is issued that causes an abend.

When the compiler is able to determine at compile time that a check will always find invalid data, a compile-time error-level message is produced and the check is removed regardless of whether MSG or ABD is in effect.

NO

No code is generated to validate data items when they are used as sending data items.

Performance considerations: NUMCHECK is much slower than NUMCHECK=NO, depending on how many zoned decimal (numeric USAGE DISPLAY) data items, packed decimal (COMP-3) data items, and binary data items are used in a COBOL program.

Since COBOL 6.2 with service applied, performance of NUMCHECK has been improved. However, performance is still best when specifying NONUMCHECK, and will be better at a higher OPT level.

ZONECHECK is deprecated and can no longer be specified in IGYCDOPT. NUMCHECK=(ZON(ALPHNUM)) gives the same results as ZONECHECK used to.

Related references

[“NUMPROC” on page 50](#)

[“TRUNC” on page 69](#)

[“ZONECHECK” on page 75](#)

[“INVDATA” on page 35](#)

NUMCLS

NUMCLS, in conjunction with NUMPROC, affects the numeric signs that the compiler treats as valid in numeric class tests.

NUMCLS specifies the sign representations that are recognized as valid by the numeric class test for data items that are defined with all of the following conditions:

- As signed (with an "S" in the PICTURE clause)
- Using DISPLAY or COMPUTATIONAL-3 (packed-decimal)
- No SEPARATE phrase on any SIGN clause

Syntax

➤ NUMCLS= 

Default

NUMCLS=PRIM

ALT

Processing with ALT accepts hexadecimal A through F as valid.

PRIM

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

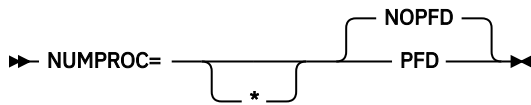
Note:

- The numeric class test is affected by how the NUMPROC and the NUMCLS options are specified.
- The NUMCLS option is effective only for NUMPROC=NOPFD. NUMPROC=PFD specifies more strict rules for valid sign configuration.

NUMPROC

NUMPROC affects the treatment and processing of signs in internal decimal and zoned decimal data.

Syntax



Default

NUMPROC=NOPFD

NOPFD

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

PFD

Optimizes the generated code, especially when a non-zero OPTIMIZE level (OPT=1 or OPT=2) is specified. 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.

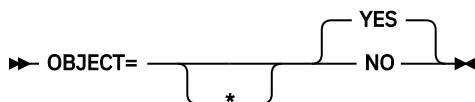
Performance consideration: Using NUMPROC=PFD generates significantly more efficient code for numeric comparisons. For most references to COMP-3 and DISPLAY numeric data items, using NUMPROC=NOPFD generates extra code because of sign "fix-up" processing. This extra code might also inhibit some other types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

Both the NUMPROC and NUMCLS options affect the numeric class test. With 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

OBJECT affects whether the generated object code is written to a file.

Syntax



Default

OBJECT=YES

YES

Places the generated object code in a file, defined by the SYSLIN DD statement, to be used as input to the binder.

NO

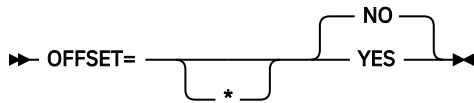
Places no object code in SYSLIN.

The OBJECT=NO option conflicts with all values for TEST other than NO.

OFFSET

OFFSET affects whether a condensed PROCEDURE DIVISION listing is produced.

Syntax



Default

OFFSET=NO

YES

Produces a condensed PROCEDURE DIVISION listing, which will contain line numbers, statement references, and the location of each block of instructions generated for a statement. The optimizer might inline paragraphs, move code around, or indeed place it after the body of the program if little used, such as the error message formatting code. As a result, there might be more than one entry in the OFFSET table of a given statement.

These items will also be written to the output listing:

- Constant area
- Program prolog areas (PPA1, PPA2, PPA3, PPA4)
- Time stamp and compiler version information
- Compiler options and program information
- Base locator table
- External symbols dictionary
- Initial heap storage maps
- Stack storage maps

NO

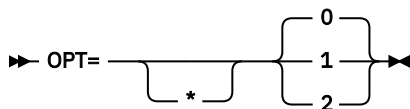
Does not condense the listing or produce the items listed above.

The LIST and OFFSET compiler options are mutually exclusive. Setting OFFSET=YES and LIST=YES results in a nonzero return code when you attempt to assemble the customization macro. For more information about conflict resolution, see [“Conflicting compiler options” on page 11](#).

OPTIMIZE

OPTIMIZE affects the level of optimization that is made to object code, and can result in performance improvements.

Syntax



Default

OPT=0

0

Specifies limited optimizations, which result in the shortest compilation time. When the TEST option is specified, full debug capabilities are available.

1

Specifies optimizations that improve application runtime performance. Optimizations at this level include basic inlining, strength reduction, simplification of complex operations into equivalent simpler operations, removal of some unreachable code and block rearrangement. Also, OPT=1 includes some

intrapblock optimizations such as common subexpression elimination and value propagation. When the TEST option is specified, most debug capabilities are available.

2

Specifies further optimizations, which include more aggressive simplifications and instruction scheduling. Also, some interblock optimizations such as global value propagation and loop invariant code motion are included. When the TEST option is specified, some debug capabilities are available.

Performance consideration: Using OPT=1 or OPT=2 generally results in more efficient runtime code.

Note:

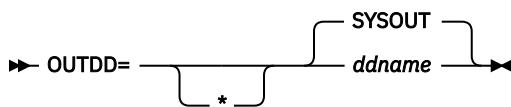
- The OPTIMIZE compiler option is fully supported for programs that use object-oriented syntax for Java interoperability.
- Optimization is set to 0 if an S-level error or U-level error occurs, or if the Program Complexity Factor exceeds the MAXPCF integer specified.

For further details, see *OPTIMIZE* in the *Enterprise COBOL Programming Guide*.

OUTDD

OUTDD specifies the ddname to which DISPLAY output should be directed.

Syntax



Default

OUTDD=SYSOUT

ddname

Specifies the ddname of the file used for runtime DISPLAY output.

Change the default for this option if, at run time, you expect there could be a conflict with another product that requires SYSOUT as a ddname.

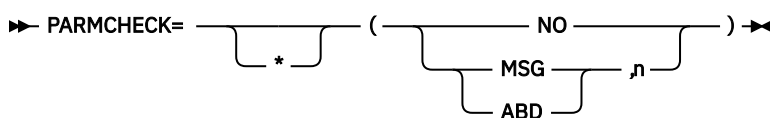
To understand how OUTDD interacts with the MSGFILE runtime option, see the description of MSGFILE in the *z/OS Language Environment Programming Reference*.

PARMCHECK

The PARMCHECK option tells the compiler to generate an extra data item following the last item in WORKING-STORAGE. This buffer data item is then used at run time to check whether a called subprogram corrupted data beyond the end of WORKING-STORAGE.

When a calling program is compiled with PARMCHECK, the compiler generates a buffer following the last data item in the WORKING-STORAGE section. At run time, before each call, the buffer is set to ALL x'AA'. After each call, the buffer is checked to see whether it was changed. The PARMCHECK option can help with your migration from COBOL 4 and earlier compilers to COBOL 6 and later compilers, and can also be used to clean up and check for good programming practices.

Syntax



Default

PARMCHECK=(NO)

MSG | ABD

Determines whether the message issued for subprogram corruption of data is a warning level message to continue processing or a terminating level message to cause an abend:

- If MSG is in effect, a runtime warning message with the name of the parameter, the line number of the CALL statement, and the program name is issued. Also, this check is done after the affected CALL statement is executed.
- If ABD is in effect, a similar message is issued, but with a terminating level that causes an abend.

n

The size in bytes of the buffer to be added after the last item in WORKING-STORAGE. Must be an integer in the range of 1 to 9999.

NO

No buffer data item is generated following the last item in WORKING-STORAGE.

Performance considerations: PARMCHECK will cause the compiler to generate slower code for programs with CALL statements. NOPARMCHECK should be in effect for good performance.

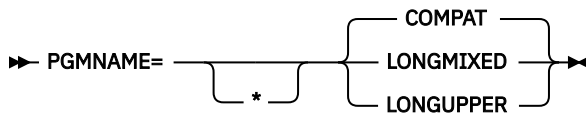
Related references

CALL statement (*Enterprise COBOL Language Reference*)

PGMNAME

PGMNAME controls the handling of program-names, entry-point names, and user-defined function-names.

Syntax



Default

PGMNAME=COMPAT

COMPAT

Program names and user-defined function names are processed in a manner compatible with COBOL/370 1.1 and VS COBOL II 1.3 and 1.4.

LONGMIXED

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

With PGMNAME (LONGMIXED), all user-defined function definitions must be specified with the AS phrase in the FUNCTION-ID paragraph. *literal-1* of the AS phrase is processed as is, without truncation, translation, or folding to uppercase.

LONGUPPER

Program names and user-defined function names are folded to uppercase by the compiler but otherwise are processed as is, without truncation or translation.

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 in the ENTRY statement
- Program-name references in:
 - CALL statements that reference nested programs, statically linked programs, or DLLs
 - SET *procedure-pointer* or *function-pointer* statements that reference statically linked programs or DLLs
 - CANCEL statements that reference nested programs
- User-defined function-names defined in the FUNCTION-ID paragraph if the AS phrase is not specified.

- *literal-1* of the AS phrase of the user-defined function-name if the AS phrase is specified.

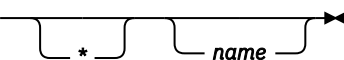
Note: If the user-defined function-name is specified with the AS phrase, *literal-1* of the AS phrase is the externalized name, not the user-defined function-name itself.

For further details, see *PGMNAME* in the *Enterprise COBOL Programming Guide*.

PRTEXIT

PRTEXIT designates a module to be called instead of output being written to the SYSPRINT data set.

Syntax

➔ PRTEXIT= 

Default

No exit is specified. Equivalent to specifying the NOPRTEXIT suboption of the EXIT compiler option. If PRTEXIT=* is coded without the *name* parameter, NOPRTEXIT cannot be overridden.

name

Identifies a module to be used with the EXIT compiler option. When the suboption for this user exit is specified, the compiler loads the named 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 information about the EXIT compiler option, see *EXIT compiler option* in the *Enterprise COBOL Programming Guide*.

QUALIFY

QUALIFY affects qualification rules and controls whether to extend qualification rules so that some data items that cannot be referenced under COBOL Standard rules can be referenced.

Syntax

➔ QUALIFY= 

Default

QUALIFY=COMPAT

COMPAT

If QUALIFY=COMPAT is in effect, the behavior will be the same as in previous COBOL compilers. A reference must be unique even if there is only one data item with exactly that complete set of qualifiers.

EXTEND

If QUALIFY=EXTEND is in effect, qualification rules are extended so that some references that are not unique by COBOL standard rules can be unique. If every level in the containing hierarchy of a group of names is qualified, the set of qualifiers is called a *complete set of qualifiers*. If there is only one data item with a specific complete set of qualifiers, the reference resolves to that data item, even if the same set of qualifiers could match with another reference as an incomplete set of qualifiers.

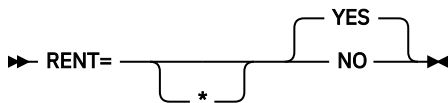
Example

```
01 A.
  02 B.
    03 C PIC X.
    02 C PIC X.
  .
  .
  .
Move space to C of A      *> Refers to 02 level C (unique only with QUALIFY(EXTEND))
Move space to C of B of A *> Refers to 03 level C (unique by COBOL standard rules)
Move space to C of B      *> Refers to 03 level C (unique by COBOL standard rules)
```

RENT

RENT affects whether generated object code is reentrant.

Syntax



Default

RENT=YES

YES

Indicates that generated object code is to be 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 generated object code is not to be reentrant.

Note:

- Compile programs with RENT if they will be run in virtual storage addresses above 16 MB.
- Execution of nonreentrant programs above 16 MB is not supported. Programs compiled with NORENT must be RMODE 24.
- The RENT compiler option is required for programs that are run under CICS.
- The LP (64) compiler option implies RENT. If the user explicitly specifies NORENT, an informational message is issued and the setting is ignored.
- The RMODE assigned to a program depends on the RENT|NORENT and RMODE compiler options. Valid combinations are shown in the following table.

Table 6. Effect of RENT and RMODE on residency mode

RENT NORENT setting	RMODE setting	Residency mode assigned
RENT	AUTO	RMODE ANY
RENT	ANY	RMODE ANY
RENT	24	RMODE 24
NORENT	AUTO	RMODE 24
NORENT	ANY	Compiler option conflict
NORENT	24	RMODE 24

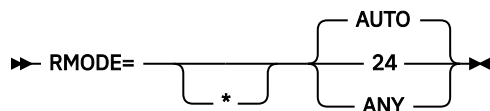
- If the THREAD compiler option is specified, the RENT compiler option must also be specified. If THREAD and NORENT are specified at the same level of precedence, the RENT option is forced on.

For further details, see *RENT* in the *Enterprise COBOL Programming Guide*.

RMODE

RMODE affects the residency mode of generated object programs.

Syntax



Default

RMODE=AUTO

24

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

ANY

Specifies that a program will have RMODE ANY if RENT is specified, and will receive an error if NORENT is specified.

AUTO

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

Note:

- 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.
- 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 are shown in the following table.

Table 7. Effect of RMODE and RENT | NORENT on residency mode

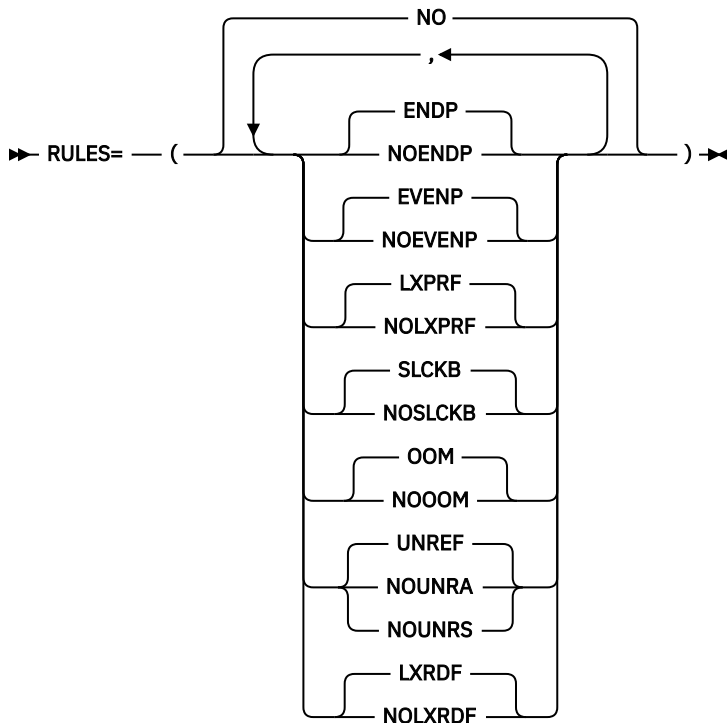
RMODE setting	RENT NORENT setting	Residency mode assigned
AUTO	RENT	RMODE ANY
AUTO	NORENT	RMODE 24
ANY	RENT	RMODE ANY
ANY	NORENT	Not applicable. A compiler error is issued.
24	RENT	RMODE 24
24	NORENT	RMODE 24

- The LP(64) compiler option implies RMODE(ANY). If the user explicitly specifies RMODE(24), an informational message is issued and the setting is ignored.

RULES

You can use the RULES option to request information about your program from the compiler to improve the program by flagging certain types of source code at compile time.

Syntax



Default

RULES = (NO)

Has the same effect as RULES=(ENDP, EVENP, LXPRF, SLCKB, OOM, UNREF, LXRDF).

You can specify the following suboptions for RULES other than default ones:

(NOENDP)

Causes the compiler to issue warning messages when the scope of a conditional statement is terminated by a period instead of an explicit scope terminator END-*.

(NOEVENP)

Causes the compiler to issue warning messages for any USAGE PACKED-DECIMAL (COMP-3) data items that have an even number of digits because those data items whose unused bits are not zero can lead to an unexpected program behavior.

Notes:

- RULES=(NOEVENP) helps you identify USAGE PACKED-DECIMAL (COMP-3) data items that have unused extra space reserved for them. However, it is not necessary to change those data items to have an odd number of digits, it is only a slightly better way of programming.
- The compiler does not issue messages for even-digit PACKED-DECIMAL data items if the name starts with DFH, DSN, EYU, or SQL. That is, data items generated for/by CICS and Db2.

(NOLXPRF)

Causes the compiler to issue warning messages for usage of inefficient COBOL features. These features might include USAGE DISPLAY numeric data items in arithmetic statements, large amounts of space padding in MOVE statements, inefficient compiler options, and other cases.

(NOSLCKB)

Causes the compiler to issue warning messages for any SYNCHRONIZED data items that cause the compiler to add slack bytes, either slack bytes within records or slack bytes between records. Each data item that causes slack bytes to be added gets a compiler diagnostic.

(NOOOM)

Causes the compiler to issue warning messages for any OCCURS DEPENDING ON clauses that are specified without *integer-1* (the minimum number of occurrences).

(NOUNRA)

When NOUNRA is specified, all level-01 and level-77 data items in the FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION, and LINKAGE SECTION that are unreferenced, including no subordinate items referenced when the item is a group, are reported, regardless of whether the definition of the data item appears directly in the user source program or was included in the program from a copy member.

(NOUNRS)

When NOUNRS is specified, all level-01 and level-77 data items in the FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION, and LINKAGE SECTION that are unreferenced, including no subordinate items referenced when the item is a group, are reported only if the definition of the data item appears directly in the user source program.

Notes:

- In COBOL, the definition of a single group item can spread across different files. When this occurs, and if the definition of the level-01 data item of the group is in the main source file, then those data items that are unreferenced will be reported when NOUNRS is in effect.
- Data items with the name prefix DFH, DSN, EYU, or SQL (that is, data items generated for/by CICS and Db2) will not be reported when NOUNRA or NOUNRS is in effect.

(NOLXRDF)

When NOLXRDF is specified, the compiler will issue warning messages when a data item is redefined by a smaller item on any level, including level-01.

Notes:

- It is not necessary to specify all of the suboptions for RULES. If a suboption is not specified, the default takes effect.
- RULES must be specified with at least one suboption for installation defaults.

SEQ

SEQ affects whether the compiler verifies that source statements are in ascending order by sequence number.

Syntax**Default**

SEQ=YES

YES

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

NO

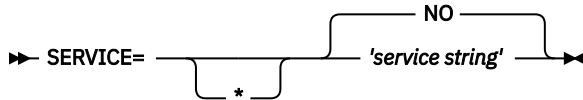
Does not perform sequence checking.

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

SERVICE

Use SERVICE to place a string in the object module if the object module is generated. If the object module is linked into a program object, the string is loaded into memory with this program object. If the Language Environment dump includes a traceback, this string is included in that traceback.

Syntax



Default

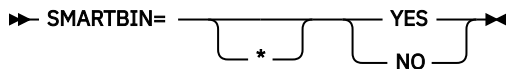
SERVICE=NO

The *service string* is limited to 64 characters in length.

SMARTBIN

Use SMARTBIN to instruct the compiler to generate modules containing additional binary metadata that enables them to be optimized by IBM Automatic Binary Optimizer (ABO) for z/OS 2.2.

Syntax



Default

SMARTBIN=YES when LP=32 is in effect.

Note: SMARTBIN is not supported when LP=64 is in effect.

The SMARTBIN=YES option enables IBM Automatic Binary Optimizer 2.2 to optimize modules created by IBM Enterprise COBOL 6.4.

When SMARTBIN=YES is in effect, the additional binary metadata is placed in a NOLOAD segment of the module. To generate the SMARTBIN metadata, compile times may increase by up to 21% at OPT=0 and 2-3% at OPT=1 and OPT=2 on an IBM z15[®] (or later) machine with zEnterprise Data Compression (zEDC) enabled (hardware compression turned on). This is in comparison to an increase of up to 33% at OPT=0 and 10% at OPT=1 and OPT=2 on an IBM z15[™] (or later) machine without zEDC enabled (hardware compression turned off). The additional metadata will also increase the size of the module on disk, requiring larger load libraries, but will not increase the size in memory when the program is running since it is not loaded. The size increase on disk will be approximately 2 times to 3 times the size of the original binary.

SMARTBIN=YES is the default when LP=32 is in effect. You can change the option to SMARTBIN=NO, however, without the additional binary metadata, COBOL modules built with Enterprise COBOL 6.4 will be ineligible for ABO optimization and you would need to recompile and test your modules in the future to maximize benefit from IBM Z[®] hardware improvements. If you use ABO or plan to in the future, the SMARTBIN=YES option is recommended.

Notes:

- IBM Automatic Binary Optimizer for z/OS 2.2 can optimize CSECTs within program modules that were generated by the following COBOL compilers:
 - Enterprise COBOL for z/OS 6
 - Newly eligible as of ABO 2.2. ABO 2.2 can only optimize COBOL 6.4 modules if the SMARTBIN=YES compiler option is in effect. ABO 2.2 can optimize COBOL 6.1, 6.2, and 6.3 modules without requiring special COBOL compiler options to be set.
 - Enterprise COBOL for z/OS 5

Newly eligible as of ABO 2.2. ABO 2.2 can optimize COBOL 5 modules without requiring special COBOL compiler options to be set.

- Enterprise COBOL for z/OS 4
- Enterprise COBOL for z/OS 3
- COBOL for OS/390® & VM 2
- COBOL for MVS™ & VM 1.2
- COBOL/370 1.1
- VS COBOL II 1.4.0
- VS COBOL II 1.3.x
- COBOL modules that have been processed by CA-Optimizer cannot be optimized by ABO. For these types of modules, it is recommended to use ABO to optimize the original module created by the COBOL compiler before it was processed by CA-Optimizer.

Refer to the [IBM Automatic Binary Optimizer for z/OS product page](#) for additional information on ABO benefits.

Related concepts

Storage and its addressability (*Enterprise COBOL Programming Guide*)

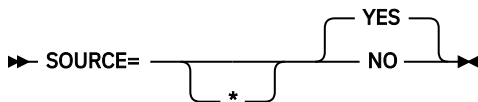
Related references

[“LP” on page 41](#)

SOURCE

SOURCE affects whether source statements are included in compiler listings.

Syntax



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. SOURCE=YES will result in the SOURCE(DEC) compiler option being in effect.

Note: You can specify SOURCE (HEX) as a compiler invocation option or in the PROCESS or CBL statement in your COBOL source program, but SOURCE=HEX cannot be specified as an installation default.

NO

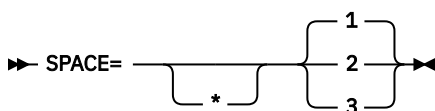
Source statements do not appear in the output.

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

SPACE

SPACE affects whether single-, double-, or triple-spacing is used in source listings.

Syntax



Default

SPACE=1

1

Provides single spacing for the source statement listing.

2

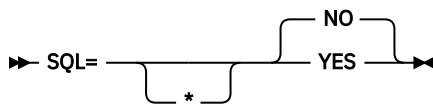
Provides double spacing for the source statement listing.

3

Provides triple spacing for the source statement listing.

SQL

SQL affects whether the Db2 coprocessor is enabled and whether Db2 options can be specified.

Syntax**Default**

SQL=NO

YES

Use to enable the Db2 coprocessor and to specify Db2 options. When the LP(64) compiler option is in effect, you must specify the SQL option if your COBOL source program contains SQL statements, and Db2 precompiler is not supported in LP(64).

The Db2 coprocessor writes the database request module (DBRM) to ddname DBRMLIB.

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 separate SQL precompiler will be used to process SQL statements before invocation of the COBOL compiler.

Note:

- You can specify the SQL option in any of the compiler option sources: compiler invocation, PROCESS or CBL statements, OPTFILE, or installation defaults.
- Use either quotation marks or apostrophes to delimit the string of Db2 options.
- Db2 options cannot be specified as part of customizing the SQL option. (Db2 options are supported only if the SQL compiler option is specified as an invocation option or in a CBL or PROCESS statement.) However, default Db2 options can be specified when you customize the Db2 product installation defaults.

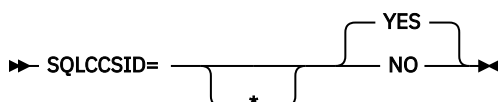
Related tasks

Compiling with the SQL option

(*Enterprise COBOL Programming Guide*)

SQLCCSID

SQLCCSID controls whether the CODEPAGE compiler option influences the processing of SQL statements when the SQL compiler option is in effect.

Syntax

Default

SQLCCSID=YES

YES

Indicates that the CODEPAGE compiler option setting will influence the processing of SQL statements within the source program when the integrated Db2 coprocessor (SQL compiler option) is used.

NO

Indicates that the CODEPAGE compiler option setting will only be used as the encoding for string literals and the COBOL application source that includes converted SQL statements. Db2 (character string) host variables will not be affected by the CODEPAGE compiler option. Instead, the encoding for Db2 (character string) host variables will come from the CCSID value found in the DSNHDECP file, which means Db2 (via DSNHDECP) determines the encoding of the Db2 data (host variables).

Notes:

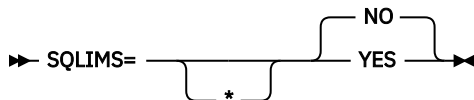
- The SQLCCSID option is supported when the LP(64) compiler option is in effect, which behaves in the same way as in LP(32).
- The SQLCCSID option has an effect only when you use the integrated Db2 coprocessor (SQL compiler option).
- The NOSQLCCSID option is recommended for applications that require the highest compatibility with the behavior of the Db2 precompiler.

Related references

SQLCCSID (*Enterprise COBOL Programming Guide*)

SQLIMS

SQLIMS affects whether the IMS SQL coprocessor is enabled and whether Information Management System (IMS) suboptions can be specified.

Syntax**Default**

SQLIMS=NO

YES

Use to enable the IMS SQL coprocessor and to specify Information Management System (IMS) suboptions. You must specify the SQLIMS option if a COBOL source program contains SQLIMS statements.

NO

Specify to have any SQLIMS statements found in the source program diagnosed and discarded.

Use SQLIMS=NO if your COBOL source programs do not contain SQLIMS statements.

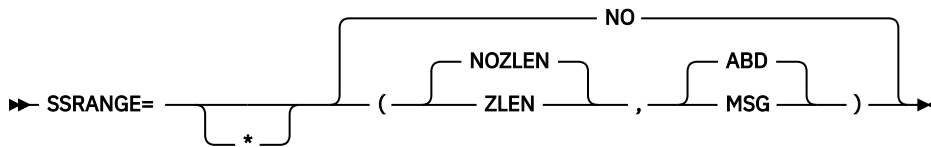
Notes:

- You can specify the SQLIMS option in any of the compiler option sources: compiler invocation, PROCESS or CBL statements, or installation defaults.
- Use either quotation marks or apostrophes to delimit the string of IMS suboptions.
- You can partition a long suboption string into multiple suboption strings in multiple CBL statements.
- When the LP(64) compiler option is in effect, the SQLIMS option is not supported. If the option is specified explicitly by the user, a diagnostic message is emitted.

SSRANGE

SSRANGE affects whether code is generated to check for out-of-range storage references.

Syntax



Default

SSRANGE=NO

ZLEN | NOZLEN

Generates code that checks subscripts, reference modifications for non-UTF-8 data items and function values, variable-length group ranges, and indexes 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 variable-length items do not exceed their defined maximum length as a result of incorrect setting of the OCCURS DEPENDING ON object. For unbounded groups or their subordinate items, checking is done only for reference modification expressions. Subscripted or indexed references to tables subordinate to an unbounded group are not checked.

The ZLEN and NOZLEN suboptions control how the compiler checks reference modification lengths:

- If ZLEN is in effect, the compiler will generate code to ensure that reference modification lengths are greater than or equal to zero. Zero-length reference modification specifications will not get an SSRANGE error at run time.
- If NOZLEN is in effect, the compiler will generate code to ensure that reference modification lengths are greater than or equal to 1. Zero-length reference modification specifications will get an SSRANGE error at run time. This is compatible with how SSRANGE behaved in previous COBOL versions.

MSG | ABD

The MSG and ABD suboptions control the runtime behavior of the COBOL program when a range check fails.

- If MSG is in effect and a range check fails, a runtime warning message will be issued. Also, the affected statements will still be executed. The program will continue and might potentially identify other out-of-range conditions.
- If ABD is in effect and a range check fails, the first out-of-range condition will result in a runtime error message and the program will abend. You can find the next potential out-of-range condition by fixing the first out-of-range condition and then recompiling and running the program again. To identify all other potential out-of-range conditions, you might need to repeat this process several times.

Performance consideration: If anything other than SSRANGE=NO is in effect at compile-time, the object code size will be increased, and the runtime overhead will also be increased to accomplish the range checking.

NO

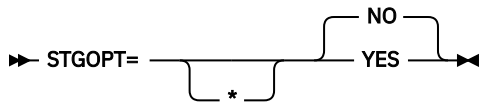
No code is generated to perform subscript or index checking at run time.

Note: If you specify anything other than SSRANGE=NO, range checks are generated by the compiler and the checks are always executed at run time. The compiled-in range checks cannot be disabled even if you specify the Language Environment runtime option CHECK(OFF).

STGOPT

The STGOPT option controls storage optimization.

Syntax



Default

STGOPT=NO

YES

If you specify STGOPT=YES, the compiler might discard any or all of the following data items, and does not allocate storage for them.

- Unreferenced LOCAL - STORAGE and WORKING - STORAGE level-77 and level-01 elementary data items
- Level-01 group items if none of their subordinate items are referenced
- Unreferenced special registers

Note: The STGOPT option is ignored for data items that have the VOLATILE clause. For details, see VOLATILE clause in the *Enterprise COBOL Language Reference*.

The compiler will not generate code to initialize discarded data items to the values in their VALUE clauses.

In addition, with STGOPT=YES, data items in the LOCAL - STORAGE SECTION can be reordered in memory to optimize performance.

NO

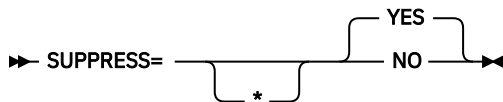
If you specify STGOPT=NO, the storage for all data items, including unreferenced data items, is allocated by the compiler, data items are never reordered to improve performance, and all data items defined with a VALUE clause are guaranteed to be initialized, even if they are unreferenced.

You can also use the RULES= (UNREF | NOUNRA | NOUNRS) option to control whether to issue warning messages for unreferenced data items. For details, see [“RULES” on page 57](#).

SUPPRESS

Use the NOSUPPRESS option to ignore the SUPPRESS phrase of all COPY statements in a program so that copybook information can appear in the listing. The copybook information can be used by debuggers, tools, and so on, without users needing to modify their source code.

Syntax



Default

SUPPRESS=YES

YES

Enables the SUPPRESS phrase of COPY statements.

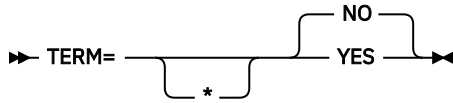
NO

Ignores the SUPPRESS phrase of COPY statements.

TERM

TERM affects whether progress and diagnostic messages are sent to the SYSTERM device.

Syntax



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.

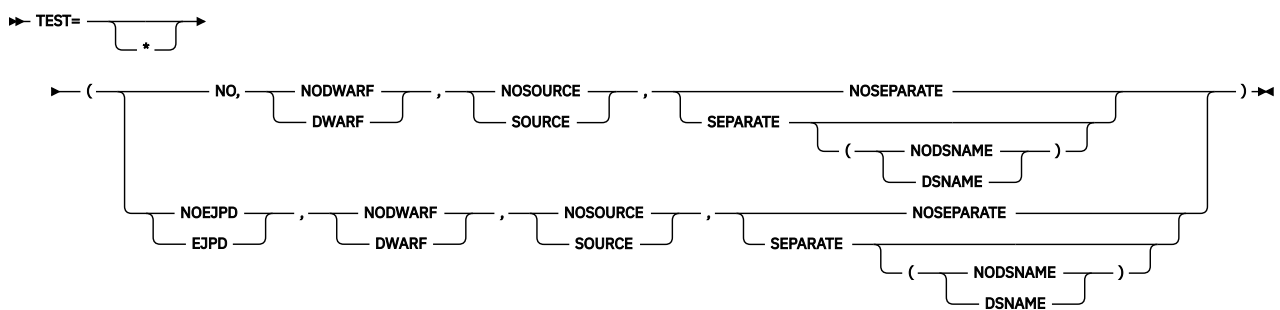
If TERM is specified in the source program, a SYSTERM DD statement must also be specified for each application program.

TERM corresponds to the TERMINAL compiler option.

TEST

TEST affects the amount of debugging information that is produced in object code, which determines the level of debugging support that is available.

Syntax



Note: Unlike specifying this option in JCL or CBL/PROCESS, all suboptions are required and must be in the order shown in the syntax diagram.

Default

TEST=(NO,NODWARF,NOSOURCE,NOSEPARATE)

(NO,DWARF,...)

If TEST=(NO,DWARF,...) is in effect, basic DWARF diagnostic information is included in the object program, or the separate debug file if SEPARATE is also in effect. This option enables application failure analysis tools, such as CEEDUMP and IBM Fault Analyzer. With TEST=(NO,DWARF,...), the debugging information is a subset of the DWARF information that is available with TEST=(DWARF,...). The DWARF diagnostic information that is produced when TEST=(NO,DWARF,...) is in effect cannot be used with IBM z/OS Debugger. Consider using TEST=(NO,DWARF,...) when use of the debugger is not needed and you want to avoid the performance implications of the TEST option while having improved usability for application failure analysis tools, such as CEEDUMP and IBM Fault Analyzer.

Debugging information generated by the compiler is in the industry-standard DWARF format. For more information about DWARF, see [About Common Debug Architecture](#) in the *DWARF/ELF Extensions Library Reference*.

(NO,NODWARF,...)

If TEST=(NO,NODWARF,...) is in effect, DWARF diagnostic information is not included in the object program, nor written to a separate debug file.

Note: SOURCE and SEPARATE are not allowed with NODWARF.

(NO,...,SOURCE,...)

If you specify TEST=(NO,...,SOURCE,...), the DWARF debugging information generated by the compiler includes the expanded source code.

Note: SOURCE is not allowed with NODWARF.

(NO,...,NOSOURCE,...)

If you specify TEST=(NO,...,NOSOURCE,...), the generated DWARF debugging information does not include the expanded source code.

(NO,...,SEPARATE(NODSNAME))

If you specify TEST=(NO,...,SEPARATE(NODSNAME)), any generated DWARF debugging information will be stored in an external file and will not cause an increase in the size of the object program. The external file name, which is the name of the SYSDEBUG dataset, used during compilation will not be stored in the object program. The default is SEPARATE(NODSNAME) when SEPARATE is specified with no suboptions.

Note: SEPARATE is not allowed with NODWARF.

(NO,...,SEPARATE(DSNAME))

If you specify TEST=(NO,...,SEPARATE(DSNAME)), any generated DWARF debugging information will be stored in an external file and will not cause an increase in the size of the object program. The external file name, which is the name of the SYSDEBUG dataset, used during compilation will be stored in the object program. This name would be used as the default at run time when DWARF information is required.

Note: SEPARATE is not allowed with NODWARF.

(NO,...,NOSEPARATE)

If you specify TEST=(NO,...,NOSEPARATE), any generated DWARF debugging information will be stored in the object program.

(other than NO)**(EJPD,...)**

If you specify TEST=(EJPD,...) and OPT=(1|2):

- The IBM z/OS Debugger commands GOTO and JUMPTO are enabled.
- Program optimization will be reduced. Optimization will be done within statements; most optimizations will not cross statement boundaries.

(NOEJPD,...)

If you specify TEST=(NOEJPD,...) and OPT=(1|2):

- The JUMPTO and GOTO commands are not enabled. However, you can still use JUMPTO and GOTO if you use the SET WARNING OFF IBM z/OS Debugger command. In this scenario, JUMPTO and GOTO will have unpredictable results.
- The normal amount of program optimization is done.

(...,DWARF,...)

If you specify TEST=(...,DWARF,...), complete DWARF diagnostic information is included in the object program, or a separate debug file, when the SEPARATE suboption is in effect. This option enables the best usability for application failure analysis tools, such as CEEDUMP and IBM Fault Analyzer.

(...,NODWARF,...)

If you specify TEST=(...,NODWARF,...), DWARF diagnostic information is not included in the object program, nor written to a separate debug file.

Note: SOURCE and SEPARATE are not allowed with NODWARF.

(...,SOURCE,...)

If you specify TEST=(. . . ,SOURCE , . . .), the DWARF debugging information generated by the compiler includes the expanded source code.

(...,NOSOURCE,...)

If you specify TEST=(. . . ,NOSOURCE , . . .), the generated DWARF debugging information does not include the expanded source code.

(...,SEPARATE(NODSNAME))

If you specify TEST=(. . . ,SEPARATE (NODSNAME)), any generated DWARF debugging information will be stored in an external file and will not cause an increase in the size of the object program. The external file name, which is the name of the SYSDEBUG dataset, used during compilation will not be stored in the object program. The default is SEPARATE (NODSNAME) when SEPARATE is specified with no suboptions.

(...,SEPARATE(DSNAME))

If you specify TEST=(. . . ,SEPARATE (DSNAME)), any generated DWARF debugging information will be stored in an external file and will not cause an increase in the size of the object program. The external file name, which is the name of the SYSDEBUG dataset, used during compilation will be stored in the object program. This name would be used as the default at run time when DWARF information is required.

Notes:

- SEPARATE is not allowed with NODWARF.
- Support for debugging DWARF debugging information in the SYSDEBUG data set with the IBM debugger requires any of the tools at the following levels:
 - IBM Debug for z Systems® 14.1 (5655-Q50) (formerly IBM Debug Tool for z/OS) or later
 - IBM Developer for z Systems 14.1 (5724-T07) or later
 - IBM Application Delivery Foundation for z Systems 3.1 (5655-AC6) or later

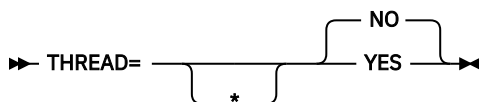
(...,NOSEPARATE)

If you specify TEST=(. . . ,NOSEPARATE), any generated DWARF debugging information will be stored in the object program.

Note: If you specify the TEST option, you must set the CODEPAGE option to the CCSID that is used for the COBOL source program. In particular, programs that use Japanese characters in DBCS literals or DBCS user-defined words must be compiled with the CODEPAGE option set to a Japanese codepage CCSID. For more information, see [“CODEPAGE” on page 20](#).

THREAD

THREAD controls whether programs are to be enabled for use in multithreaded applications.

Syntax**Default**

THREAD=NO

YES

Use YES to indicate that programs are to be enabled for execution in Language Environment enclaves that have multiple POSIX threads or PL/I tasks.

NO

Use NO to indicate that programs are not to be enabled for execution in Language Environment enclaves that have multiple POSIX threads or PL/I tasks.

Performance consideration: If the THREAD compiler option is specified, runtime performance might be degraded because of the serialization logic that is automatically generated.

Note:

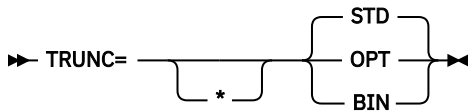
- The THREAD compiler option is ignored when LP(64) is in effect. If the user explicitly specifies the THREAD option, an informational message is issued.
- If the THREAD compiler option is specified, the program is enabled for use in a threaded application. However, THREAD can be used in nonthreaded 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.
- If 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, RENT is forced on.
- For COBOL programs to run in a threaded application, all COBOL programs in the run unit must be compiled with the THREAD compiler option.
- If 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:
 - ALTER statement
 - DEBUG-ITEM special register
 - GO TO statement without a procedure name
 - INITIAL phrase in the PROGRAM-ID paragraph
 - Nested programs
 - RERUN
 - Segmentation module
 - MERGE or Format 1 SORT statements
 - STOP *literal* statement
 - USE FOR DEBUGGING statement
- If you compile programs with the THREAD compiler option, the following special registers are allocated upon each invocation:
 - ADDRESS OF
 - JSON-CODE
 - JSON-STATUS
 - RETURN-CODE
 - SORT-CONTROL
 - SORT-CORE-SIZE
 - SORT-FILE-SIZE
 - SORT-MESSAGE
 - SORT-MODE-SIZE
 - SORT-RETURN
 - TALLY
 - XML-CODE
 - XML-EVENT
 - XML-INFORMATION
 - XML-NAMESPACE
 - XML-NAMESPACE-PREFIX
 - XML-NNAMESPACE
 - XML-NNAMESPACE-PREFIX

- XML-NTEXT
- XML-TEXT

TRUNC

TRUNC affects the way that binary data is truncated during MOVE and arithmetic operations.

Syntax



Default

TRUNC=STD

BIN

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

- Output binary fields are truncated only at halfword, fullword, and doubleword boundaries, rather than at PICTURE-clause limits.
- Sending binary fields are treated as halfwords, fullwords, or doublewords, and no assumption is made that the values are limited to those implied by the PICTURE clause.
- DISPLAY converts and outputs the full content of binary fields with no truncation to the PICTURE description.

Note: When TRUNC (BIN) and NUMCHECK (BIN) are both in effect and an error message or an abend is generated, if you intend to switch to TRUNC (STD | OPT) later for better performance, you must correct the data; if not, you can turn off NUMCHECK (BIN) to reduce the execution time of the application and avoid an error message or an abend.

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.

STD

Conforms to the 85 COBOL 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.

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.

Recommendations:

- TRUNC=BIN is the recommended option for programs that use binary values set by other products. Other products, such as IMS, Db2, C/C++, FORTRAN, and PL/I, might place values in COBOL binary data items that do not conform to the PICTURE clause of those data items.
- You can use TRUNC=OPT with CICS programs provided that your data conforms to the PICTURE clause for the binary data items.

- Setting this option affects program runtime 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.

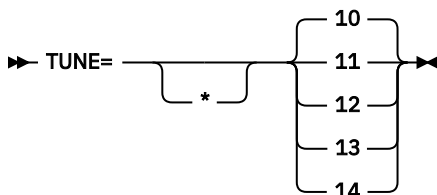
Related references

[“NUMCHECK” on page 45](#)

TUNE

The TUNE option specifies the architecture for which the executable program will be optimized.

Syntax



Default

The default TUNE level matches the ARCH level if ARCH is specified. If ARCH is not specified, both ARCH and TUNE default to 10.

Current supported architecture levels and groups of models are as follows:

10

Generates code that is optimized for the 2827-xxx (IBM zEnterprise EC12) and 2828-xxx (IBM zEnterprise BC12) models in IBM z/Architecture mode.

11

Generates code that is optimized for the 2964-xxx (IBM z13) and 2965-xxx (IBM z13s) models in IBM z/Architecture mode.

12

Generates code that is optimized for the 3906-xxx (IBM z14) and 3907-xxx (IBM z14 ZR1) models in IBM z/Architecture mode.

13

Generates code that is optimized for the 8561-xxx (IBM z15) and 8562-xxx (IBM z15 T02) models in IBM z/Architecture mode.

14

Generates code that is optimized for the 3931-xxx (IBM z16) model in IBM z/Architecture mode.

The TUNE option specifies the architecture for which the executable program will be optimized. The TUNE level controls how the compiler selects the available machine instructions, while staying within the restrictions of the ARCH level in effect. The TUNE option does so to provide the highest performance possible on the given TUNE architecture, choosing from the instructions allowed by the given ARCH level.

Note: TUNE impacts performance only; it does not impact the processor model on which you will be able to run your application.

Select TUNE to match the architecture of the machine where your application will run most often. The TUNE level must always be greater or equal to the ARCH level because you will want to tune an application for a machine on which it can run. The compiler enforces this by adjusting TUNE up rather than ARCH down. TUNE does not specify where an application can run and it affects optimization only.

For example, if you have production machines that are z15 and disaster recovery (DR) machines that are z14, you should compile with ARCH=12 and TUNE=13. In this way, your programs will run well on the DR machines and run as fast as possible in production.

Note: If the specified TUNE level is lower than the specified ARCH level, the compiler adjusts the TUNE level to match the ARCH level.

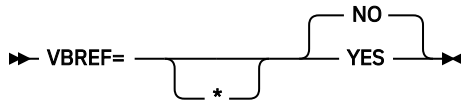
Related references

[“ARCH” on page 16](#)

VBREF

VBREF affects whether a cross-reference of statements to line numbers and a statement-usage summary is produced.

Syntax



Default

VBREF=NO

YES

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

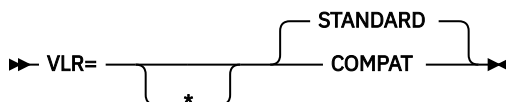
NO

Does not produce a cross-reference or statement-summary listing.

VLR

The VLR option affects the file status returned from READ statements for variable-length records when the length of record returned is inconsistent with the record descriptions. It eases your migration from earlier versions to Enterprise COBOL 6, if your programs have READ statements that result in a record length conflict.

Syntax



Default

VLR=STANDARD

After the execution of a READ statement:

- If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for the file, the portion of the record area that is to the right of the last valid character read is undefined.
- If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for the file, the record is truncated on the right to the maximum size.

In either of these cases, the READ statement is successful, and the file status is set to either 00 (hiding the record length conflict condition) or 04 (indicating that a record length conflict has occurred), depending on the VLR compiler option setting.

COMPAT

VLR=COMPAT checks the size of the read record against the "VERYING IN SIZE FROM min TO max" declaration of the FD clause. If you specify VLR=COMPAT, you get the status value of 00 when READ statements encounter a record length conflict.

Note: This setting can hide I/O problems that can arise with the wrong length read situation. Use the VLR=COMPAT option with caution, and check for correct READ statements.

STANDARD

VLR=STANDARD checks the size of the read record against the declaration of the FD level 01 clause. If you specify VLR=STANDARD, you get the status value of 04 when READ statements encounter a record length conflict.

You can add code to test for FS=04 to avoid accessing undefined data in a record and also avoid getting protection exceptions for attempting to reference a part of the record that was truncated.

The following example shows how VLR option checks the size of the read record against the declaration of the FD clause and FD level 01 clause.

```
FD MYDD
  block contains 0 records
  record varying in size from 10 to 80
  recording mode V.
01 REC-20
  02 PIC X(20).
01 REC-50.
  02 PIC X(50).
```

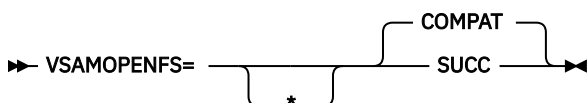
Table 8. Length of record read and file status

Length of record read	=5 < min varying < min level 01	= 15 >= min varying < min level 01	= 40 >= min varying <= max varying >= min level 01 <= max level 01	= 70 >= min varying <= max varying > max level 01
COBOL 4	FS=04	FS=00	FS=00	FS=00
COBOL 6 VLR (COMPAT)	FS=04	FS=00	FS=00	FS=00
COBOL 6 VLR (STANDARD)	FS=04	FS=04	FS=00	FS=04

VSAMOPENFS

The VSAMOPENFS option affects the user file status reported from successful VSAM OPEN statements that require verified file integrity check.

Syntax



Default

VSAMOPENFS=COMPAT

COMPAT

If you specify VSAMOPENFS=COMPAT, the statement returns the file status 97 when a VSAM OPEN statement is successfully verified. This is compatible with pre-COBOL 6 runtime behavior.

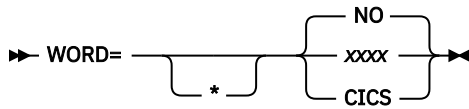
SUCC

If you specify VSAMOPENFS=SUCC, the statement returns the file status 00 when a VSAM OPEN statement is successfully verified. This allows users to simply check for 0 in the first digit of the returned file status, as they usually do with other successful operations.

WORD

WORD indicates which alternate reserved-word table is to be used during compilation.

Syntax



Default

WORD=NO

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 8](#).

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 (\$).

- CBE
- DGEN
- DIAG
- DMAP
- DOPT
- ECWI
- FGEN
- INIT
- LIBO
- LIBR
- LSTR
- LVL0
- LVL1
- LVL2
- LVL3
- LVL8
- OSCN
- PGEN
- RCTL
- RDPR
- RDSC
- RWT
- SAW
- SCAN
- SIMD
- XREF

NO

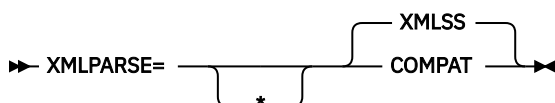
Indicates that no alternative reserved word table is to be used as the default.

Note:

- Specification of WORD affects the interpretation of input reserved words. System names (such as UPSI and SYSPUNCH) and the 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.
- Changing the default value of the WORD=XXXX option conflicts with all values for FLAGSTD other than NO.

XMLPARSE

XMLPARSE indicates which parser is to be used for processing XML input, and therefore which XML parsing capabilities are available.

Syntax**Default**

XMLPARSE=XMLSS

COMPAT

XML PARSE statements are processed using the XML parser that is a built-in component of the COBOL library. The XML PARSE statement results and operational behaviors are then compatible with those obtained with Enterprise COBOL 3, and also with COBOL 4 when XMLPARSE(COMPAT) was used.

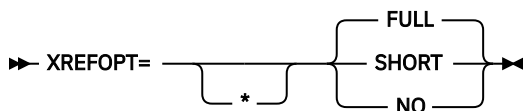
XMLSS

XML PARSE statements are processed using the z/OS XML System Services parser. The following XML parsing capabilities are available only when this suboption is in effect:

- Namespace processing enhancements
- The ENCODING, RETURNING NATIONAL, and VALIDATING phrases of the XML PARSE statement
- Support for direct parsing of XML documents encoded in UTF-8
- Support for parsing very large XML documents, a buffer of XML at a time
- Offloading of XML parsing to System z® Application Assist Processors (zAAPs)

XREFOPT

XREFOPT sets the default value for the XREF compiler option, which affects the amount of cross-reference information produced in listings.

Syntax**Default**

XREFOPT=FULL

FULL

Produces a sorted cross-reference of the names used in the program, and indicates the lines where the names are defined. Also produces a COPY/BASIS cross-reference. If SOURCE=YES is also specified, embedded cross-reference information is printed on the same lines as the source.

SHORT

Produces a sorted listing of only the explicitly referenced variables, and produces a COPY/BASIS cross-reference.

NO

Suppresses the cross-reference listings.

Note:

- XREFOPT=NO conflicts with values of DBCSXREF other than NO.
- It is recommended that you not change the default to XREFOPT=NO. If XREFOPT=NO, the COPY/BASIS cross-reference might in some cases be incomplete or missing.

For further details, see *XREF* in the *Enterprise COBOL Programming Guide*.

ZONECHECK

ZONECHECK is deprecated and can no longer be specified in IGYCDOPT. NUMCHECK=(ZON(ALPHNUM)) gives the same results as ZONECHECK used to.

For details, see [“NUMCHECK” on page 45](#).

Related references

[“NUMCHECK” on page 45](#)

[“NUMPROC” on page 50](#)

[“INVDATA” on page 35](#)

ZONEDATA

The ZONEDATA option tells the compiler whether the data in USAGE DISPLAY and PACKED-DECIMAL data items is valid, and if not, what the behavior of the compiler should be.

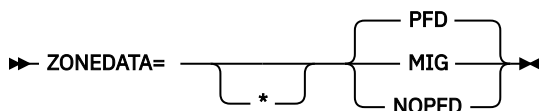
Since most users have valid data in their USAGE DISPLAY and USAGE PACKED-DECIMAL data items, most users should use ZONEDATA=PFD. Even if you find that your programs are processing invalid data at run time (using the NUMCHECK compiler option), you should change your programs to avoid processing invalid data, and use ZONEDATA=PFD.

Note: The ZONEDATA option is deprecated but is tolerated for compatibility, and it is replaced by the INVDATA option.

When the ZONEDATA option is specified, it is mapped to the equivalent INVDATA option as follows:

```
ZONEDATA=PFD -> NOINVDATA
ZONEDATA=NOPFD -> INVDATA(NOFORCENUMCMP,CLEANSIGN)
ZONEDATA=MIG -> INVDATA(FORCENUMCMP,CLEANSIGN)
```

Syntax



Default

ZONEDATA=PFD

PFD

When ZONEDATA=PFD is in effect, the compiler assumes that all data in USAGE DISPLAY and PACKED-DECIMAL data items is valid, and generates the most efficient code possible to make numeric comparisons. For example, the compiler might generate a string comparison to avoid numeric conversion.

MIG

When ZONEDATA=MIG is in effect, the compiler generates instructions to do numeric comparisons that ignore the zone bits of each digit in zoned decimal data items. For example, the zoned decimal value is converted to packed-decimal with a pack instruction before the comparison. The compiler will also avoid performing known optimizations that might produce a different result than COBOL 4

(or earlier versions) when a zoned decimal or packed decimal data item has invalid digits or an invalid sign code, or when a zoned decimal data item has invalid zone bits.

NOPFD

When ZONEDATA=NOPFD is in effect, the compiler generates instructions for numeric comparisons or an alphanumeric comparison of zoned decimal data in the same manner as COBOL 4 (or earlier versions) does when using NUMPROC=NOPFD | PFD with COBOL 4 (or earlier versions):

- In the cases where COBOL 4 (or earlier versions) considered the zone bits, the compiler generates an alphanumeric comparison which will also consider the zone bits of each digit in zoned decimal data items. The zoned decimal value remains as zoned decimal.
- In the cases where COBOL 4 ignored the zone bits, the compiler generates numeric comparisons that ignore the zone bits of each digit in zoned decimal data items. The zoned decimal value is converted to packed-decimal with a PACK instruction before the comparison.

In order for the compiler to handle zone bits in the same way as COBOL 4 (or earlier versions) did when generating comparisons of zoned decimal data, the NUMPROC suboption used in COBOL 6 must match the NUMPROC suboption used in COBOL 4 (or earlier versions):

- To get the COBOL 4 (or earlier versions) NUMPROC=NOPFD behavior in COBOL 6, use ZONEDATA=NOPFD and NUMPROC=NOPFD in COBOL 6.
- To get the COBOL 4 (or earlier versions) NUMPROC=PFD behavior in COBOL 6, use ZONEDATA=NOPFD and NUMPROC=PFD in COBOL 6.

The compiler will also avoid performing known optimizations that might produce a different result than COBOL 4 (or earlier versions) when a zoned decimal or packed decimal data item has invalid digits or an invalid sign code, or when a zoned decimal data item has invalid zone bits.

Note: The sign code must be a valid sign code according to the NUMPROC compiler option setting. In addition, the low-order byte must have a valid zone (x ' F ') for unsigned and signed with either SIGN IS LEADING or SIGN IS SEPARATE.

In all, to ease your migration to COBOL 6:

- If your digits, sign codes, and zone bits are valid, use ZONEDATA=PFD and the same NUMPROC setting that you used with COBOL 4 when using COBOL 6.
- If you have invalid digits, invalid sign codes, or invalid zone bits in your data, change your programs or systems so that your programs do not have invalid data in numeric data items at run time.

Once you have corrected your programs or systems, you can use the preferred ZONEDATA=PFD option. Only if you cannot contain this work and must continue to run with invalid data, consider the following choices for ZONEDATA:

- If you used NUMPROC=MIG with COBOL 4, use ZONEDATA=MIG and NUMPROC=NOPFD with COBOL 6.
- If you used NUMPROC=NOPFD with COBOL 4, use ZONEDATA=NOPFD and NUMPROC=NOPFD with COBOL 6.
- If you used NUMPROC=PFD with COBOL 4, use ZONEDATA=NOPFD and NUMPROC=PFD with COBOL 6.

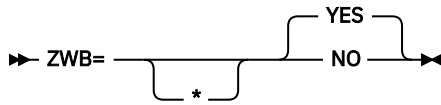
Note: It is not always possible to entirely match the behavior of the old compiler even with these options when faced with clearly invalid data. For example, even for compares, ZONEDATA=NOPFD isn't going to give the same result in all cases as COBOL 4.

Performance consideration: ZONEDATA=PFD gives better runtime performance than ZONEDATA=NOPFD | MIG does. ZONEDATA=NOPFD | MIG disables some of the optimizations that NUMPROC=PFD can give.

ZWB

ZWB determines whether the compiler removes the sign from signed zoned decimal fields before they are compared to alphanumeric fields at run time.

Syntax



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.

Note:

- Setting this option affects program runtime 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.
- Application programmers use ZWB=NO to test input numeric fields for SPACES.

Chapter 3. Customizing Enterprise COBOL

The following sections describe the supplied user jobs and modules that you can modify to customize 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.

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.V6R4M0.SIGYSAMP. However, these sample modification jobs are not customized for your particular system. You must customize them.

Table 9 on page 79 shows the names of the sample modification jobs, which are described in detail in the referenced information.

Copy members from IGY.V6R4M0.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.

Table 9. Summary of user modification jobs for Enterprise COBOL

Description	Job	See:
Changing the compiler options default module	IGYWDOPT	“Changing the compiler options default module” on page 80
Overriding options specified as fixed	IGYWUOPT	“Overriding options specified as fixed” on page 81
Changing reserved words	IGYWRWD	“Changing reserved words” on page 81

Changing the defaults for compiler options

You can change the defaults for compiler options, and can override fixed compiler options.

To change the defaults for compiler options:

1. Copy the source of options module IGYCDOPT from IGY.V6R4M0.SIGYSAMP into the appropriate job, IGYWDOPT or IGYWUOPT, in place of the two-line comment.
2. Change the parameters on the IGYCOPT macro call to match the compiler options that you have selected for your installation.

Observe the following requirements when coding the changed IGYCOPT macro call:

- Place continuation character (X in the source) 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 the macro.
- Specify options and suboptions in uppercase. Only suboptions that are strings can be specified in mixed case or lowercase.
- 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 quotation marks ("). A null string can be specified with either contiguous apostrophes or quotation marks.

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 is properly parsed if this error is committed.

- Code only those options whose default value you want to change. The IGYCOPT macro generates the default value for any option that you do not code.

For a worksheet to help you plan your default compiler options, see [“IGYCDOPT worksheet for compiler options” on page 2](#). For descriptions of the options, see [Chapter 2, “Enterprise COBOL compiler options,” on page 11](#).

- Place an END statement after the macro instruction.

Note: When the assembler encounters an unknown keyword on any macro call, like the call to IGYCOPT in `igycopt.asm`, it treats the keyword as a positional parameter. Because the current version of the IGYCOPT macro did not check for positional parameters, those unknown keywords were completely ignored. Now, the IGYCOPT macro has been changed so that it detects these positional parameters and emits appropriate error messages. If you rebuild an existing customization macro that used to assemble `RC=00`, it might fail with assembly errors due to either unknown compiler options that were previously treated as positional parameters and thus ignored, or compiler options coded with incorrect syntax such as `OPTION()` instead of `OPTION=` in the COBOL customization macro. You must remove or correct these unknown or improperly coded options and rebuild the IGYCDOPT module.

For additional details about how to code macro calls, see the *High Level Assembler for z/OS Language Reference*.

Changing the compiler options default module

To change the defaults for the Enterprise COBOL compiler options, modify the sample job IGYWDOPT.

To choose default values, use the information in [Chapter 2, “Enterprise COBOL compiler options,” on page 11](#).

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.V6R4M0.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.

Overriding options specified as fixed

Occasionally, you might have an application that needs to override one or more options that were specified as fixed.

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.V6R4M0.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.V6R4M0.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 DSNNAME=YOURLIB in the SYSLMOD DD statement to the name of the data set that you want your IGYCDOPT module bound 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 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.

Changing reserved words

To change the words that Enterprise COBOL treats as reserved, use the reserved word table utility.

The reserved words used by Enterprise COBOL 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 8). You can change the reserved words by using the reserved word table utility (IGY8RWTU) to modify IGYCRWT or IGYCCICS, or by creating additional reserved word tables. You can also modify tables that you previously created.

The reserved word table utility accepts control statements 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 words to be flagged with an informational message whenever they are used in a program. To produce these information messages, you must modify the IGYCRWT reserved word table and compile using the FLAGSTD option.
- Add words to be flagged with a severe 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 character strings that cannot be used, see [“WORD” on page 73](#).

At compile time, the value of the compiler option `WORD(yyyy)` identifies the reserved word table to be used. `yyyy` is the unique 1- to 4-character identification that you specified in the member name `IGYCyyyy`. 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.

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
```

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 the 85 COBOL 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
```

Creating or modifying a reserved word table

You can create or modify a reserved word table using one of these existing tables:

- Member `IGY8RWRD` in `IGY.V6R4M0.SIGYSAMP` (the IBM-supplied default reserved word table)
- Member `IGY8CICS` in `IGY.V6R4M0.SIGYSAMP` (the IBM-supplied CICS reserved word table)

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

The source file of the existing reserved word table above should have four parts: Parts I, II, III, and IV. Follow these steps to modify the file and non-SMP/E JCL:

1. Make a private copy of the latest `IGY8RWRD` or `IGY8CICS`.

Note: `IGY8RWRD` or `IGY8CICS` might be updated with the addition of new reserved words either through a new release or continuous delivery PTFs. Be sure to rebuild your custom reserved word table using the latest version of `IGY8RWRD` or `IGY8CICS` as the basis.

2. Skip Part I (all lines up to and including the line with the keyword `MOD`). Make *no* alterations in this part of the file!
3. Edit Part II by placing asterisks in column 1 of the lines that contain 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 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 in [“Coding control statements”](#) on page 83.
6. Modify and run the JCL, as described in [“Running JCL that creates a reserved word table”](#) on page 86. You also must create a unique 1- to 4-character identification for the new reserved word table and supply it in the JCL. Your user-defined reserved word table will be a program object in a PDSE dataset.

Coding control statements

To create or modify a reserved word table, you must understand the syntax of the control statements that affect them.

The following figure illustrates the format for coding reserved word 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 control statements

As shown in the preceding figure, the 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 and the FLAGSTD compiler option is in effect

RSTR

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

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

You need to follow the rules when you code your control statements.

These rules are:

- 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

This topic 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

Follow the rules when you code the control statement operands.

These rules are:

- 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.

ABBR statement

The ABBR statement defines an alternative symbol for the reserved word specified. The symbol can be used when you code a program.

ABBR reserved-word: user-word [comments]

Note:

- The user-word becomes a reserved word and can be used in place of the reserved-word specified in this statement.
- The reserved-word remains a reserved word with its original definition.
- The source listing shows the original source—the symbol as you coded it.
- 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  REDEFINES: REDEF  
SEPARATE:  SEP
```

INFO statement

The INFO statement specifies the COBOL words that are to be flagged by the compiler, and it can also be used to control the use of nested programs.

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

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 if the FLAGSTD compiler option is in effect.

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.

RSTR statement

The RSTR statement specifies COBOL words that cannot be used in a program, and it can also be used to control the use of nested programs.

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

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.

The following reserved words can be restricted only by using option 1:

DATA
ENVIRONMENT
ID
IDENTIFICATION
PROGRAM-ID
WORKING-STORAGE
LOCAL-STORAGE
LINKAGE

Modifying and running non-SMP/E JCL

Use sample job IGYWRWD in IGY.V6R4M0.SIGYSAMP to create or modify a reserved word table.

The sample job uses a member based on IGY8RWRD or IGY8CICS in IGY.V6R4M0.SIGYSAMP as the input to the reserve word utility. It creates program object IGYCxxxx (where xxxx is the user identification) in IGY.V6R4M0.SIGYCOMP.

To modify the JCL for IGYWRWD, do these steps:

1. Modify the job statement for your site.
2. Add JES ROUTE records if required.
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. To point to your modified reserved word table, do *only* one of the following steps:
 - Change the data set name in //RSWDREAD DD DSN=... to the data set name and member name of your modified reserved word table.
 - Replace the RSWDREAD DD with //RSWDREAD DD * and insert your modified reserved table immediately following that line.

For specific instructions, see the comments in job IGYWRWD.

5. Change the name of the data set in 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 in 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 in the SYSLMOD DD statement.

After you run IGYWRWD, if 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.

Running JCL that creates a reserved word table

The JCL that creates a reserved word table contains STEP1, STEP2, and STEP3.

The three steps do the following tasks, respectively:

1. Run the reserved word table utility with your modified table.
2. Assemble your modified reserved word table.
3. Produce a runtime program object from the object module.

After you run the job, a 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.

Tailoring the cataloged procedures to your site

You might want to tailor the cataloged procedures IGYWC, IGYWCL, or IGYWCLG for use at your site.

Consider these changes:

- Modifying the data set name prefixes if you used a different prefix than the IBM-supplied ones for Enterprise COBOL or Language Environment target data sets
- Removing the STEPLIB DD statements if you placed IGY.V6R4M0.SIGYCOMP, CEE.SCEERUN, and CEE.SCEERUN2 in the LNKST 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

Appendix A. Accessibility features for Enterprise COBOL for z/OS

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

Accessibility features

z/OS includes the following major accessibility features:

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

z/OS uses the latest W3C Standard, [WAI-ARIA 1.0](http://www.w3.org/TR/wai-aria/) (<http://www.w3.org/TR/wai-aria/>), to ensure compliance to [US Section 508](https://www.access-board.gov/ict/) (<https://www.access-board.gov/ict/>) and [Web Content Accessibility Guidelines \(WCAG\) 2.0](http://www.w3.org/TR/WCAG20/) (<http://www.w3.org/TR/WCAG20/>). To take advantage of accessibility features, use the latest release of your screen reader in combination with the latest web browser that is supported by this product.

Keyboard navigation

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

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

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

- *z/OS TSO/E Primer* (<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4p120>)
- *z/OS TSO/E User's Guide* (<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4c240/APPENDIX1.3>)
- *z/OS ISPF User's Guide Volume I* (<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ispzug70>)
- *IBM Developer for z/OS Knowledge Center* (http://www.ibm.com/support/knowledgecenter/SSQ2R2/rdz_welcome.html?lang=en)

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

Interface information

The Enterprise COBOL for z/OS online product documentation is available in IBM Knowledge Center, which is viewable from a standard web browser.

PDF files have limited accessibility support. With PDF documentation, you can use optional font enlargement, high-contrast display settings, and can navigate by keyboard alone.

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

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

Related accessibility information

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service
800-IBM-3383 (800-426-3383)
(within North America)

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see [IBM Accessibility](http://www.ibm.com/able) (www.ibm.com/able).

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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
U.S.A.

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want 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:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
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 document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1996, 2024.

PRIVACY POLICY CONSIDERATIONS:

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, or to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> in the section entitled "Cookies, Web Beacons and Other Technologies,"

and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

Programming interface information

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



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

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms might or might not have the same meaning in other languages.

This glossary includes terms and definitions from the following publications:

- *ANSI INCITS 23-1985, Programming languages - COBOL*, as amended by *ANSI INCITS 23a-1989, Programming Languages - COBOL - Intrinsic Function Module for COBOL*, and *ANSI INCITS 23b-1993, Programming Languages - Correction Amendment for COBOL*
- *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*
- *ANSI X3.172-2002, American National Standard Dictionary for Information Systems*
- *INCITS/ISO/IEC 1989-2002, Information technology - Programming languages - COBOL*
- *INCITS/ISO/IEC 1989:2014, Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

American National Standard definitions are preceded by an asterisk (*).

A

* abbreviated combined relation condition

The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

abend

Abnormal termination of a program.

above the 2 GB bar

Storage located above the so-called 2 GB bar (or boundary). This storage is only addressable by AMODE 64 programs.

above the 16 MB line

Storage located above the so-called 16 MB line (or boundary) but below the 2 GB bar. This storage is not addressable by AMODE 24 programs. Before IBM introduced the MVS/XA architecture in the 1980s, the virtual storage for a program was limited to 16 MB. Programs that have been link-edited as AMODE 24 can address only 16 MB of space, as though they were kept under an imaginary storage line. Since VS COBOL II, a program can have AMODE 31 and can be loaded above the 16 MB line.

* access mode

The manner in which records are to be operated upon within a file.

* actual decimal point

The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

actual document encoding

For an XML document, one of the following encoding categories that the XML parser determines by examining the first few bytes of the document:

- ASCII
- EBCDIC
- UTF-8
- UTF-16, either big-endian or little-endian
- Other unsupported encoding
- No recognizable encoding

*** alphabet-name**

A user-defined word, in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, that assigns a name to a specific character set or collating sequence or both.

*** alphabetic character**

A letter or a space character.

alphanumeric character position

See *character position*.

alphabetic data item

A data item that is described with a PICTURE character string that contains only the symbol A. An alphabetic data item has USAGE DISPLAY.

*** alphanumeric character**

Any character in the single-byte character set of the computer.

alphanumeric data item

A general reference to a data item that is described implicitly or explicitly as USAGE DISPLAY, and that has category alphanumeric, alphanumeric-edited, or numeric-edited.

alphanumeric-edited data item

A data item that is described by a PICTURE character string that contains at least one instance of the symbol A or X and at least one of the simple insertion symbols B, 0, or /. An alphanumeric-edited data item has USAGE DISPLAY.

*** alphanumeric function**

A function whose value is composed of a string of one or more characters from the alphanumeric character set of the computer.

alphanumeric group item

A group item that is defined without a GROUP-USAGE NATIONAL clause. For operations such as INSPECT, STRING, and UNSTRING, an alphanumeric group item is processed as though all its content were described as USAGE DISPLAY regardless of the actual content of the group. For operations that require processing of the elementary items within a group, such as MOVE CORRESPONDING, ADD CORRESPONDING, or INITIALIZE, an alphanumeric group item is processed using group semantics.

alphanumeric literal

A literal that has an opening delimiter from the following set: ' , " , X ' , X " , Z ' , or Z " . The string of characters can include any character in the character set of the computer.

*** alternate record key**

A key, other than the prime record key, whose contents identify a record within an indexed file.

ANSI (American National Standards Institute)

An organization that consists of producers, consumers, and general-interest groups and establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

argument

(1) An identifier, a literal, an arithmetic expression, or a function-identifier that specifies a value to be used in the evaluation of a function. (2) An operand of the USING phrase of a CALL or INVOKE statement, used for passing values to a called program or an invoked method.

*** arithmetic expression**

A numeric literal, an identifier representing a numeric elementary item, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

*** arithmetic operation**

The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

*** arithmetic operator**

A single character, or a fixed two-character combination that belongs to the following set:

Character	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

*** arithmetic statement**

A statement that causes an arithmetic operation to be executed. The arithmetic statements are ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT.

array

An aggregate that consists of data objects, each of which can be uniquely referenced by subscripting. An array is roughly analogous to a COBOL table.

*** ascending key**

A key upon the values of which data is ordered, starting with the lowest value of the key up to the highest value of the key, in accordance with the rules for comparing data items.

ASCII

American National Standard Code for Information Interchange. The standard code uses a coded character set that is based on 7-bit coded characters (8 bits including parity check). The standard is used for information interchange between data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

IBM has defined an extension to ASCII (characters 128-255).

ASCII DBCS

See *double-byte ASCII*.

assignment-name

A name that identifies the organization of a COBOL file and the name by which it is known to the system.

*** assumed decimal point**

A decimal point position that does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

AT END condition

A condition that is caused during the execution of a READ, RETURN, or SEARCH statement under certain conditions:

- A READ statement runs on a sequentially accessed file when no next logical record exists in the file, or when the number of significant digits in the relative record number is larger than the size of the relative key data item, or when an optional input file is not available.
- A RETURN statement runs when no next logical record exists for the associated sort or merge file.
- A SEARCH statement runs when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

B

basic character set

The basic set of characters used in writing words, character-strings, and separators of the language. The basic character set is implemented in single-byte EBCDIC. The extended character set includes DBCS characters, which can be used in comments, literals, and user-defined words.

Synonymous with *COBOL character set* in the 85 COBOL Standard.

batch compilation

Synonymous with *sequence of programs*.

big-endian

The default format that the mainframe and the AIX® workstation use to store binary data and UTF-16 characters. In this format, the least significant byte of a binary data item is at the highest address and the least significant byte of a UTF-16 character is at the highest address. Compare with *little-endian*.

binary item

A numeric data item that is represented in binary notation (on the base 2 numbering system). The decimal equivalent consists of the decimal digits 0 through 9, plus an operational sign. The leftmost bit of the item is the operational sign.

binary search

A dichotomizing search in which, at each step of the search, the set of data elements is divided by two; some appropriate action is taken in the case of an odd number.

*** block**

A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block can contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical records that are either contained within the block or that overlap the block. Synonymous with *physical record*.

boolean condition

A boolean condition determines whether a boolean literal is true or false. A boolean condition can only be used in a constant conditional expression.

boolean literal

Can be either B'1', indicating a true value, or B'0', indicating a false value. Boolean literals can only be used in constant conditional expressions.

breakpoint

A place in a computer program, usually specified by an instruction, where external intervention or a monitor program can interrupt the program as it runs.

buffer

A portion of storage that is used to hold input or output data temporarily.

built-in function

See *intrinsic function*.

business method

A method of an enterprise bean that implements the business logic or rules of an application. (Oracle)

byte

A string that consists of a certain number of bits, usually eight, treated as a unit, and representing a character or a control function.

byte order mark (BOM)

A Unicode character that can be used at the start of UTF-16 or UTF-32 text to indicate the byte order of subsequent text; the byte order can be either big-endian or little-endian.

bytecode

Machine-independent code that is generated by the Java compiler and executed by the Java interpreter. (Oracle)

C**callable services**

In Language Environment, a set of services that a COBOL program can invoke by using the conventional Language Environment-defined call interface. All programs that share the Language Environment conventions can use these services.

called program

A program that is the object of a CALL statement. At run time the called program and calling program are combined to produce a *run unit*.

*** calling program**

A program that executes a CALL to another program.

canonical decomposition

A way to represent a single precomposed Unicode character using two or more Unicode characters. A canonical decomposition is typically used to separate latin letters with a diacritical mark so that the latin letter and the diacritical mark are represented individually. See *precomposed character* for an example showing a precomposed Unicode character and its canonical decomposition.

case structure

A program-processing logic in which a series of conditions is tested in order to choose between a number of resulting actions.

cataloged procedure

A set of job control statements that are placed in a partitioned data set called the procedure library (SYS1.PROCLIB). You can use cataloged procedures to save time and reduce errors in coding JCL.

CCSID

See *coded character set identifier*.

century window

A 100-year interval within which any two-digit year is unique. Several types of century window are available to COBOL programmers:

- For the windowing intrinsic functions DATE-T0-YYYYMMDD, DAY-T0-YYYYDDD, and YEAR-T0-YYYY, you specify the century window with *argument-2*.
- For Language Environment callable services, you specify the century window in CEESSEN.

*** character**

The basic indivisible unit of the language.

character encoding unit

A unit of data that corresponds to one code point in a coded character set. One or more character encoding units are used to represent a character in a coded character set. Also known as *encoding unit*.

For USAGE NATIONAL, a character encoding unit corresponds to one 2-byte code point of UTF-16.

For USAGE DISPLAY, a character encoding unit corresponds to a byte.

For USAGE DISPLAY-1, a character encoding unit corresponds to a 2-byte code point in the DBCS character set.

character position

The amount of physical storage or presentation space required to hold or present one character. The term applies to any class of character. For specific classes of characters, the following terms apply:

- *Alphanumeric character position*, for characters represented in USAGE DISPLAY
- *DBCS character position*, for DBCS characters represented in USAGE DISPLAY-1
- *National character position*, for characters represented in USAGE NATIONAL; synonymous with *character encoding unit* for UTF-16

character set

A collection of elements that are used to represent textual information, but for which no coded representation is assumed. See also *coded character set*.

character string

A sequence of contiguous characters that form a COBOL word, a literal, a PICTURE character string, or a comment-entry. A character string must be delimited by separators.

checkpoint

A point at which information about the status of a job and the system can be recorded so that the job step can be restarted later.

*** class**

The entity that defines common behavior and implementation for zero, one, or more objects. The objects that share the same implementation are considered to be objects of the same class. Classes can be defined hierarchically, allowing one class to inherit from another.

class (object-oriented)

The entity that defines common behavior and implementation for zero, one, or more objects. The objects that share the same implementation are considered to be objects of the same class.

*** class condition**

The proposition (for which a truth value can be determined) that the content of an item is wholly alphabetic, is wholly numeric, is wholly DBCS, is wholly Kanji, or consists exclusively of the characters that are listed in the definition of a class-name.

*** class definition**

The COBOL source unit that defines a class.

class hierarchy

A tree-like structure that shows relationships among object classes. It places one class at the top and one or more layers of classes below it. Synonymous with *inheritance hierarchy*.

*** class identification entry**

An entry in the CLASS-ID paragraph of the IDENTIFICATION DIVISION; this entry contains clauses that specify the class-name and assign selected attributes to the class definition.

class-name (object-oriented)

The name of an object-oriented COBOL class definition.

*** class-name (of data)**

A user-defined word that is defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION; this word assigns a name to the proposition (for which a truth value can be defined) that the content of a data item consists exclusively of the characters that are listed in the definition of the class-name.

class object

The runtime object that represents a class.

*** clause**

An ordered set of consecutive COBOL character strings whose purpose is to specify an attribute of an entry.

client

In object-oriented programming, a program or method that requests services from one or more methods in a class.

COBOL character set

The set of characters used in writing COBOL syntax. The complete COBOL character set consists of these characters:

Character	Meaning
0,1, . . . ,9	Digit
A,B, . . . ,Z	Uppercase letter
a,b, . . . ,z	Lowercase letter
	Space
+	Plus sign
-	Minus sign (hyphen)
*	Asterisk
/	Slant (forward slash)
=	Equal sign
\$	Currency sign
,	Comma
;	Semicolon
.	Period (decimal point, full stop)

Character	Meaning
"	Quotation mark
'	Apostrophe
(Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than
:	Colon
_	Underscore

*** COBOL word**

See *word*.

code page

An assignment of graphic characters and control function meanings to all code points. For example, one code page could assign characters and meanings to 256 code points for 8-bit code, and another code page could assign characters and meanings to 128 code points for 7-bit code. For example, one of the IBM code pages for English on the workstation is IBM-1252 and on the host is IBM-1047. A *coded character set*.

code point

A unique bit pattern that is defined in a coded character set (code page). Graphic symbols and control characters are assigned to code points.

coded character set

A set of unambiguous rules that establish a character set and the relationship between the characters of the set and their coded representation. Examples of coded character sets are the character sets as represented by ASCII or EBCDIC code pages or by the UTF-16 encoding scheme for Unicode.

coded character set identifier (CCSID)

An IBM-defined number in the range 1 to 65,535 that identifies a specific code page.

*** collating sequence**

The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, comparing, and for processing indexed files sequentially.

*** column**

A byte position within a print line or within a reference format line. The columns are numbered from 1, by 1, starting at the leftmost position of the line and extending to the rightmost position of the line. A column holds one single-byte character.

*** combined condition**

A condition that is the result of connecting two or more conditions with the AND or the OR logical operator. See also *condition* and *negated combined condition*.

combining characters

A Unicode character used to modify other succeeding or preceding Unicode characters. Combining characters are typically Unicode diacritical mark used to modify latin letters. See *precomposed character* for an example of combining character U+0308 (¨) used with latin letter U+0061 (a).

*** comment-entry**

An entry in the IDENTIFICATION DIVISION that is used for documentation and has no effect on execution.

comment line

A source program line represented by an asterisk (*) in the indicator area of the line or by an asterisk followed by greater-than sign (*>) as the first character string in the program text area (Area A plus Area B), and any characters from the character set of the computer that follow in Area A and Area B of that line. A comment line serves only for documentation. A special form of comment line represented

by a slant (/) in the indicator area of the line and any characters from the character set of the computer in Area A and Area B of that line causes page ejection before the comment is printed.

*** common program**

A program that, despite being directly contained within another program, can be called from any program directly or indirectly contained in that other program.

compilation group

Synonymous with *sequence of programs*.

compilation unit

A unit of COBOL source code that can be separately compiled: a program, class, user-defined function, or prototype definition. Also known as a source unit.

compilation variable

A symbolic name for a particular literal value or the value of a compile-time arithmetic expression as specified by the DEFINE directive or by the DEFINE compiler option.

*** compile**

(1) To translate a program expressed in a high-level language into a program expressed in an intermediate language, assembly language, or a computer language. (2) To prepare a machine-language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler.

*** compile time**

The time at which COBOL source code is translated, by a COBOL compiler, to a COBOL object program.

compile-time arithmetic expression

A subset of arithmetic expressions that are specified in the DEFINE and EVALUATE directives or in a constant conditional expression. The difference between compile-time arithmetic expressions and regular arithmetic expressions is that in a compile-time arithmetic expression:

- The exponentiation operator shall not be specified.
- All operands shall be integer numeric literals or arithmetic expressions in which all operands are integer numeric literals.
- The expression shall be specified in such a way that a division by zero does not occur.

compiler

A program that translates source code written in a higher-level language into machine-language object code.

compiler-directing statement

A statement that causes the compiler to take a specific action during compilation. The standard compiler-directing statements are COPY, REPLACE, and USE.

*** complex condition**

A condition in which one or more logical operators act upon one or more conditions. See also *condition*, *negated simple condition*, and *negated combined condition*.

complex ODO

Certain forms of the OCCURS DEPENDING ON clause:

- Variably located item or group: A data item described by an OCCURS clause with the DEPENDING ON option is followed by a nonsubordinate data item or group. The group can be an alphanumeric group or a national group.
- Variably located table: A data item described by an OCCURS clause with the DEPENDING ON option is followed by a nonsubordinate data item described by an OCCURS clause.
- Table with variable-length elements: A data item described by an OCCURS clause contains a subordinate data item described by an OCCURS clause with the DEPENDING ON option.
- Index name for a table with variable-length elements.
- Element of a table with variable-length elements.

component

(1) A functional grouping of related files. (2) In object-oriented programming, a reusable object or program that performs a specific function and is designed to work with other components and applications. JavaBeans is Oracle's architecture for creating components.

composed form

Representation of a precomposed Unicode character through a canonical decomposition. See *precomposed character* for details.

*** computer-name**

A system-name that identifies the computer where the program is to be compiled or run.

condition (exception)

An exception that has been enabled, or recognized, by Language Environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware or the operating system and result in an interrupt. They can also be detected by language-specific generated code or language library code.

condition (expression)

A status of data at run time for which a truth value can be determined. Where used in this information in or in reference to "condition" (*condition-1*, *condition-2*, . . .) of a general format, the term refers to a conditional expression that consists of either a simple condition optionally parenthesized or a combined condition (consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses) for which a truth value can be determined. See also *simple condition*, *complex condition*, *negated simple condition*, *combined condition*, and *negated combined condition*.

*** conditional expression**

A simple condition or a complex condition specified in an EVALUATE, IF, PERFORM, or SEARCH statement. See also *simple condition* and *complex condition*.

*** conditional phrase**

A phrase that specifies the action to be taken upon determination of the truth value of a condition that results from the execution of a conditional statement.

*** conditional statement**

A statement that specifies that the truth value of a condition is to be determined and that the subsequent action of the object program depends on this truth value.

*** conditional variable**

A data item one or more values of which has a condition-name assigned to it.

*** condition-name**

A user-defined word that assigns a name to a subset of values that a conditional variable can assume; or a user-defined word assigned to a status of an implementor-defined switch or device.

*** condition-name condition**

The proposition (for which a truth value can be determined) that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

*** CONFIGURATION SECTION**

A section of the ENVIRONMENT DIVISION that describes overall specifications of source and object programs and class definitions.

CONSOLE

A COBOL environment-name associated with the operator console.

constant conditional expression

A subset of conditional expressions that may be used in IF directives or WHEN phrases of the EVALUATE directives.

A constant conditional expression shall be one of the following items:

- A relation condition in which both operands are literals or arithmetic expressions that contain only literal terms. The condition shall follow the rules for relation conditions, with the following additions:
 - The operands shall be of the same category. An arithmetic expression is of the category numeric.

- If literals are specified and they are not numeric literals, the relational operator shall be “IS EQUAL TO”, “IS NOT EQUAL TO”, “IS =”, “IS NOT =”, or “IS <>”.

See also *relation condition*.

- A defined condition. See also *defined condition*.
- A boolean condition. See also *boolean condition*.
- A complex condition formed by combining the above forms of simple conditions into complex conditions by using AND, OR, and NOT. Abbreviated combined relation conditions shall not be specified. See also *complex condition*.

contained program

A COBOL program that is nested within another COBOL program.

*** contiguous items**

Items that are described by consecutive entries in the DATA DIVISION, and that bear a definite hierarchic relationship to each other.

copybook

A file or library member that contains a sequence of code that is included in the source program at compile time using the COPY statement. The file can be created by the user, supplied by COBOL, or supplied by another product. Synonymous with *copy file*.

*** counter**

A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

cross-reference listing

The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

currency-sign value

A character string that identifies the monetary units stored in a numeric-edited item. Typical examples are \$, USD, and EUR. A currency-sign value can be defined by either the CURRENCY compiler option or the CURRENCY SIGN clause in the SPECIAL -NAMES paragraph of the ENVIRONMENT DIVISION. If the CURRENCY SIGN clause is not specified and the NOCURRENCY compiler option is in effect, the dollar sign (\$) is used as the default currency-sign value. See also *currency symbol*.

currency symbol

A character used in a PICTURE clause to indicate the position of a currency sign value in a numeric-edited item. A currency symbol can be defined by either the CURRENCY compiler option or the CURRENCY SIGN clause in the SPECIAL -NAMES paragraph of the ENVIRONMENT DIVISION. If the CURRENCY SIGN clause is not specified and the NOCURRENCY compiler option is in effect, the dollar sign (\$) is used as the default currency sign value and currency symbol. Multiple currency symbols and currency sign values can be defined. See also *currency sign value*.

*** current record**

In file processing, the record that is available in the record area associated with a file.

*** current volume pointer**

A conceptual entity that points to the current volume of a sequential file.

D

*** data clause**

A clause, appearing in a data description entry in the DATA DIVISION of a COBOL program, that provides information describing a particular attribute of a data item.

*** data description entry**

An entry in the DATA DIVISION of a COBOL program that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

DATA DIVISION

The division of a COBOL program or method that describes the data to be processed by the program or method: the files to be used and the records contained within them; internal WORKING-STORAGE

records that will be needed; data to be made available in more than one program in the COBOL run unit.

*** data item**

A unit of data (excluding literals) defined by a COBOL program or by the rules for function evaluation.

data set

Synonym for *file*.

*** data-name**

A user-defined word that names a data item described in a data description entry. When used in the general formats, data-name represents a word that must not be reference-modified, subscripted, or qualified unless specifically permitted by the rules for the format.

DBCS

See *double-byte character set (DBCS)*.

DBCS character

Any character defined in IBM's double-byte character set.

DBCS character position

See *character position*.

DBCS data item

A data item that is described by a PICTURE character string that contains at least one symbol G, or, when the NSYMBOL (DBCS) compiler option is in effect, at least one symbol N. A DBCS data item has USAGE DISPLAY-1.

*** debugging line**

Any line with a D in the indicator area of the line.

*** debugging section**

A section that contains a USE FOR DEBUGGING statement.

*** declarative sentence**

A compiler-directing sentence that consists of a single USE statement terminated by the separator period.

*** declaratives**

A set of one or more special-purpose sections, written at the beginning of the PROCEDURE DIVISION, the first of which is preceded by the key word DECLARATIVE and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler-directing sentence, followed by a set of zero, one, or more associated paragraphs.

*** de-edit**

The logical removal of all editing characters from a numeric-edited data item in order to determine the unedited numeric value of the item.

defined condition

A compile-time condition that tests whether a compilation variable is defined. Defined conditions are specified in IF directives or WHEN phrases of the EVALUATE directives.

*** delimited scope statement**

Any statement that includes its explicit scope terminator.

*** delimiter**

A character or a sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

dependent region

In IMS, the MVS virtual storage region that contains message-driven programs, batch programs, or online utilities.

*** descending key**

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

digit

Any of the numerals from 0 through 9. In COBOL, the term is not used to refer to any other symbol.

*** digit position**

The amount of physical storage required to store a single digit. This amount can vary depending on the usage specified in the data description entry that defines the data item.

*** direct access**

The facility to obtain data from storage devices or to enter data into a storage device in such a way that the process depends only on the location of that data and not on a reference to data previously accessed.

display floating-point data item

A data item that is described implicitly or explicitly as USAGE DISPLAY and that has a PICTURE character string that describes an external floating-point data item.

*** division**

A collection of zero, one, or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four divisions in a COBOL program: Identification, Environment, Data, and Procedure.

*** division header**

A combination of words followed by a separator period that indicates the beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
PROCEDURE DIVISION.
```

DLL

See *dynamic link library (DLL)*.

DLL application

An application that references imported programs, functions, or variables.

DLL linkage

A CALL in a program that has been compiled with the DLL and NODYNAM options; the CALL resolves to an exported name in a separate module, or to an INVOKE of a method that is defined in a separate module.

do construct

In structured programming, a DO statement is used to group a number of statements in a procedure. In COBOL, an inline PERFORM statement functions in the same way.

do-until

In structured programming, a do-until loop will be executed at least once, and until a given condition is true. In COBOL, a TEST AFTER phrase used with the PERFORM statement functions in the same way.

do-while

In structured programming, a do-while loop will be executed if, and while, a given condition is true. In COBOL, a TEST BEFORE phrase used with the PERFORM statement functions in the same way.

document type declaration

An XML element that contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or DTD.

document type definition (DTD)

The grammar for a class of XML documents. See *document type declaration*.

double-byte ASCII

An IBM character set that includes DBCS and single-byte ASCII characters. (Also known as ASCII DBCS.)

double-byte EBCDIC

An IBM character set that includes DBCS and single-byte EBCDIC characters. (Also known as EBCDIC DBCS.)

double-byte character set (DBCS)

A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

DWARF

DWARF was developed by the UNIX International Programming Languages Special Interest Group (SIG). It is designed to meet the symbolic, source-level debugging needs of different languages in a unified fashion by supplying language-independent debugging information. A DWARF file contains debugging data organized into different elements. For more information, see [*DWARF program information*](#) in the *DWARF/ELF Extensions Library Reference*.

*** dynamic access**

An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement.

dynamic CALL

A CALL *literal* statement in a program that has been compiled with the DYNAM option and the NODLL option, or a CALL *identifier* statement in a program that has been compiled with the NODLL option.

dynamic-length

An adjective describing an item whose logical length might change at runtime.

dynamic-length elementary item

An elementary data item whose data declaration entry contains the DYNAMIC LENGTH clause.

dynamic-length group

A group item that contains a subordinate dynamic-length elementary item.

dynamic link library (DLL)

A file that contains executable code and data that are bound to a program at load time or run time, rather than during linking. Several applications can share the code and data in a DLL simultaneously. Although a DLL is not part of the executable file for a program, it can be required for an executable file to run properly.

dynamic storage area (DSA)

Dynamically acquired storage composed of a register save area and an area available for dynamic storage allocation (such as program variables). A DSA is allocated upon invocation of a program or function and persists for the duration of the invocation instance. DSAs are generally allocated within stack segments managed by Language Environment.

E*** EBCDIC (Extended Binary-Coded Decimal Interchange Code)**

A coded character set based on 8-bit coded characters.

EBCDIC character

Any one of the symbols included in the EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set.

EBCDIC DBCS

See *double-byte EBCDIC*.

edited data item

A data item that has been modified by suppressing zeros or inserting editing characters or both.

*** editing character**

A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
	Space
0	Zero
+	Plus
-	Minus
CR	Credit
DB	Debit
Z	Zero suppress
*	Check protect
\$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
/	Slant (forward slash)

EGCS

See *extended graphic character set (EGCS)*.

EJB

See *Enterprise JavaBeans*.

EJB container

A container that implements the EJB component contract of the J2EE architecture. This contract specifies a runtime environment for enterprise beans that includes security, concurrency, life cycle management, transaction, deployment, and other services. An EJB container is provided by an EJB or J2EE server. (Oracle)

EJB server

Software that provides services to an EJB container. An EJB server can host one or more EJB containers. (Oracle)

element (text element)

One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type.

*** elementary item**

A data item that is described as not being further logically subdivided.

encapsulation

In object-oriented programming, the technique that is used to hide the inherent details of an object. The object provides an interface that queries and manipulates the data without exposing its underlying structure. Synonymous with *information hiding*.

enclave

When running under Language Environment, an enclave is analogous to a run unit. An enclave can create other enclaves by using LINK and by using the system() function in C.

encoding unit

See *character encoding unit*.

end class marker

A combination of words, followed by a separator period, that indicates the end of a COBOL class definition. The end class marker is:

```
END CLASS class-name.
```


end method marker

A combination of words, followed by a separator period, that indicates the end of a COBOL method definition. The end method marker is:

```
END METHOD method-name.
```

*** end of PROCEDURE DIVISION**

The physical position of a COBOL source program after which no further procedures appear.

*** end program marker**

A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program marker is:

```
END PROGRAM program-name.
```

enterprise bean

A component that implements a business task and resides in an EJB container. (Oracle)

Enterprise JavaBeans

A component architecture defined by Oracle for the development and deployment of object-oriented, distributed, enterprise-level applications.

*** entry**

Any descriptive set of consecutive clauses terminated by a separator period and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, or DATA DIVISION of a COBOL program.

*** environment clause**

A clause that appears as part of an ENVIRONMENT DIVISION entry.

ENVIRONMENT DIVISION

One of the four main component parts of a COBOL program, class definition, or method definition. The ENVIRONMENT DIVISION describes the computers where the source program is compiled and those where the object program is run. It provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

environment-name

A name, specified by IBM, that identifies system logical units, printer and card punch control characters, report codes, program switches or all of these. When an environment-name is associated with a mnemonic-name in the ENVIRONMENT DIVISION, the mnemonic-name can be substituted in any format in which such substitution is valid.

environment variable

Any of a number of variables that define some aspect of the computing environment, and are accessible to programs that operate in that environment. Environment variables can affect the behavior of programs that are sensitive to the environment in which they operate.

escape sequence

A sequence of characters that are used to represent certain special characters within string literals and character literals.

Escape sequences consist of two or more characters, the first of which is the backslash (\) character, which is called the "escape character"; the remaining characters determine the interpretation of the escape sequence. For example, \n is an escape sequence that denotes a newline character.

Escape sequences are used in programming languages such as C, C++, Java, or Python. COBOL does not have the concept of "escape sequence" or "escape character". To handle special characters within COBOL literals, see *Basic alphanumeric literals* and *DBCS literals* in the *Enterprise COBOL for z/OS Language Reference*.

execution time

See *run time*.

execution-time environment

See *runtime environment*.

*** explicit scope terminator**

A reserved word that terminates the scope of a particular PROCEDURE DIVISION statement.

exponent

A number that indicates the power to which another number (the base) is to be raised. Positive exponents denote multiplication; negative exponents denote division; and fractional exponents denote a root of a quantity. In COBOL, an exponential expression is indicated with the symbol ** followed by the exponent.

*** expression**

An arithmetic or conditional expression.

*** extend mode**

The state of a file after execution of an OPEN statement, with the EXTEND phrase specified for that file, and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

extended graphic character set (EGCS)

A graphic character set, such as a kanji character set, that requires two bytes to identify each graphic character. It is refined and replaced by *double-byte character set (DBCS)*.

Extensible Markup Language

See *XML*.

extensions

COBOL syntax and semantics supported by IBM compilers in addition to those described in the 85 COBOL Standard.

external code page

For XML documents, the value specified by the CODEPAGE compiler option.

*** external data**

The data that is described in a program as external data items and external file connectors.

*** external data item**

A data item that is described as part of an external record in one or more programs of a run unit and that can be referenced from any program in which it is described.

*** external data record**

A logical record that is described in one or more programs of a run unit and whose constituent data items can be referenced from any program in which they are described.

external decimal data item

See *zoned decimal data item* and *national decimal data item*.

*** external file connector**

A file connector that is accessible to one or more object programs in the run unit.

external floating-point data item

See *display floating-point data item* and *national floating-point data item*.

external program

The outermost program. A program that is not nested.

*** external switch**

A hardware or software device, defined and named by the implementor, which is used to indicate that one of two alternate states exists.

F

factory data

Data that is allocated once for a class and shared by all instances of the class. Factory data is declared in the WORKING-STORAGE SECTION of the DATA DIVISION in the FACTORY paragraph of the class definition, and is equivalent to Java private static data.

factory method

A method that is supported by a class independently of an object instance. Factory methods are declared in the FACTORY paragraph of the class definition, and are equivalent to Java public static methods. They are typically used to customize the creation of objects.

*** figurative constant**

A compiler-generated value referenced through the use of certain reserved words.

*** file**

A collection of logical records.

*** file attribute conflict condition**

An unsuccessful attempt has been made to execute an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

*** file clause**

A clause that appears as part of any of the following DATA DIVISION entries: file description entry (FD entry) and sort-merge file description entry (SD entry).

*** file connector**

A storage area that contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

File-Control

The name of an ENVIRONMENT DIVISION paragraph in which the data files for a given source program are declared.

file control block

Block containing the addresses of I/O routines, information about how they were opened and closed, and a pointer to the file information block.

*** file control entry**

A SELECT clause and all its subordinate clauses that declare the relevant physical attributes of a file.

FILE-CONTROL paragraph

A paragraph in the ENVIRONMENT DIVISION in which the data files for a given source unit are declared.

*** file description entry**

An entry in the FILE SECTION of the DATA DIVISION that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

*** file-name**

A user-defined word that names a file connector described in a file description entry or a sort-merge file description entry within the FILE SECTION of the DATA DIVISION.

*** file organization**

The permanent logical file structure established at the time that a file is created.

file position indicator

A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that an optional input file is not available, or that the AT END condition already exists, or that no valid next record has been established.

*** FILE SECTION**

The section of the DATA DIVISION that contains file description entries and sort-merge file description entries together with their associated record descriptions.

file system

The collection of files that conform to a specific set of data-record and file-description protocols, and a set of programs that manage these files.

*** fixed file attributes**

Information about a file that is established when a file is created and that cannot subsequently be changed during the existence of the file. These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the code set, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

*** fixed-length record**

A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of bytes.

fixed-point item

A numeric data item defined with a PICTURE clause that specifies the location of an optional sign, the number of digits it contains, and the location of an optional decimal point. The format can be either binary, packed decimal, or external decimal.

floating comment indicators (*>)

A floating comment indicator indicates a comment line if it is the first character string in the program-text area (Area A plus Area B), or indicates an inline comment if it is after one or more character strings in the program-text area.

floating point

A format for representing numbers in which a real number is represented by a pair of distinct numerals. In a floating-point representation, the real number is the product of the fixed-point part (the first numeral) and a value obtained by raising the implicit floating-point base to a power denoted by the exponent (the second numeral). For example, a floating-point representation of the number 0.0001234 is 0.1234 -3, where 0.1234 is the mantissa and -3 is the exponent.

floating-point data item

A numeric data item that contains a fraction and an exponent. Its value is obtained by multiplying the fraction by the base of the numeric data item raised to the power that the exponent specifies.

*** format**

A specific arrangement of a set of data.

*** function**

A temporary data item whose value is determined at the time the function is referenced during the execution of a statement.

*** function-identifier**

A syntactically correct combination of character strings and separators that references a function. The data item represented by a function is uniquely identified by a function-name with its arguments, if any. A function-identifier can include a reference-modifier. A function-identifier that references an alphanumeric function can be specified anywhere in the general formats that an identifier can be specified, subject to certain restrictions. A function-identifier that references an integer or numeric function can be referenced anywhere in the general formats that an arithmetic expression can be specified.

function-name

A word that names the mechanism whose invocation, along with required arguments, determines the value of a function.

function-pointer data item

A data item in which a pointer to an entry point can be stored. A data item defined with the USAGE IS FUNCTION-POINTER clause contains the address of a function entry point. Typically used to communicate with C and Java programs.

G**garbage collection**

The automatic freeing by the Java runtime system of the memory for objects that are no longer referenced.

*** global name**

A name that is declared in only one program but that can be referenced from the program and from any program contained within the program. Condition-names, data-names, file-names, record-names, report-names, and some special registers can be global names.

global reference

A reference to an object that is outside the scope of a method.

group item

(1) A data item that is composed of subordinate data items. See *alphanumeric group item* and *national group item*. (2) When not qualified explicitly or by context as a national group or an alphanumeric group, the term refers to groups in general.

grouping separator

A character used to separate units of digits in numbers for ease of reading. The default is the character comma.

H**header label**

(1) A data-set label that precedes the data records in a unit of recording media. (2) Synonym for *beginning-of-file label*.

hide (a method)

To redefine (in a subclass) a factory or static method defined with the same method-name in a parent class. Thus, the method in the subclass *hides* the method in the parent class.

*** high-order end**

The leftmost character of a string of characters.

hiperspace

In a z/OS environment, a range of up to 2 GB of contiguous virtual storage addresses that a program can use as a buffer.

I**IBM COBOL extension**

COBOL syntax and semantics supported by IBM compilers in addition to those described in the 85 COBOL Standard.

IDENTIFICATION DIVISION

One of the four main component parts of a COBOL program, class definition, or method definition. The IDENTIFICATION DIVISION identifies the program, class, or method. The IDENTIFICATION DIVISION can include the following documentation: author name, installation, or date.

*** identifier**

A syntactically correct combination of character strings and separators that names a data item. When referencing a data item that is not a function, an identifier consists of a data-name, together with its qualifiers, subscripts, and reference-modifier, as required for uniqueness of reference. When referencing a data item that is a function, a function-identifier is used.

IGZCBSN

The bootstrap routine for COBOL/370 1.1. It must be link-edited with any module that contains a COBOL/370 1.1 program.

IGZCBSO

The bootstrap routine for COBOL for MVS & VM 1.2, COBOL for OS/390 & VM and Enterprise COBOL. It must be link-edited with any module that contains a COBOL for MVS & VM 1.2, COBOL for OS/390 & VM or Enterprise COBOL program.

IGZEBST

The bootstrap routine for VS COBOL II. It must be link-edited with any module that contains a VS COBOL II program.

ILC

InterLanguage Communication. Interlanguage communication is defined as programs that call or are called by other high-level languages. Assembler is not considered a high-level language; thus, calls to and from assembler programs are not considered ILC.

*** imperative statement**

A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement can consist of a sequence of imperative statements.

*** implicit scope terminator**

A separator period that terminates the scope of any preceding unterminated statement, or a phrase of a statement that by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

IMS

Information Management System, IBM licensed product. IMS supports hierarchical databases, data communication, translation processing, and database backout and recovery.

*** index**

A computer storage area or register, the content of which represents the identification of a particular element in a table.

*** index data item**

A data item in which the values associated with an index-name can be stored in a form specified by the implementor.

indexed data-name

An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

*** indexed file**

A file with indexed organization.

*** indexed organization**

The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

indexing

Synonymous with *subscripting* using index-names.

*** index-name**

A user-defined word that names an index associated with a specific table.

inheritance

A mechanism for using the implementation of a class as the basis for another class. By definition, the inheriting class conforms to the inherited classes. Enterprise COBOL does not support *multiple inheritance*; a subclass has exactly one immediate superclass.

inheritance hierarchy

See *class hierarchy*.

*** initial program**

A program that is placed into an initial state every time the program is called in a run unit.

*** initial state**

The state of a program when it is first called in a run unit.

inline

In a program, instructions that are executed sequentially, without branching to routines, subroutines, or other programs.

inline comments

An inline comment is identified by a floating comment indicator (*>) preceded by one or more character-strings in the program-text area, and can be written on any line of a compilation group. All characters that follow the floating comment indicator up to the end of area B are comment text.

*** input file**

A file that is opened in the input mode.

*** input mode**

The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

*** input-output file**

A file that is opened in the I-O mode.

*** INPUT-OUTPUT SECTION**

The section of the ENVIRONMENT DIVISION that names the files and the external media required by an object program or method and that provides information required for transmission and handling of data at run time.

*** input-output statement**

A statement that causes files to be processed by performing operations on individual records or on the file as a unit. The input-output statements are ACCEPT (with the identifier phrase), CLOSE, DELETE, DISPLAY, OPEN, READ, REWRITE, SET (with the TO ON or TO OFF phrase), START, and WRITE.

*** input procedure**

A set of statements, to which control is given during the execution of a format 1 SORT statement, for the purpose of controlling the release of specified records to be sorted.

instance data

Data that defines the state of an object. The instance data introduced by a class is defined in the WORKING-STORAGE SECTION of the DATA DIVISION in the OBJECT paragraph of the class definition. The state of an object also includes the state of the instance variables introduced by classes that are inherited by the current class. A separate copy of the instance data is created for each object instance.

*** integer**

(1) A numeric literal that does not include any digit positions to the right of the decimal point. (2) A numeric data item defined in the DATA DIVISION that does not include any digit positions to the right of the decimal point. (3) A numeric function whose definition provides that all digits to the right of the decimal point are zero in the returned value for any possible evaluation of the function.

integer function

A function whose category is numeric and whose definition does not include any digit positions to the right of the decimal point.

Interactive System Productivity Facility (ISPF)

An IBM software product that provides a menu-driven interface for the TSO or VM user. ISPF includes library utilities, a powerful editor, and dialog management.

interlanguage communication (ILC)

The ability of routines written in different programming languages to communicate. ILC support lets you readily build applications from component routines written in a variety of languages.

intermediate result

An intermediate field that contains the results of a succession of arithmetic operations.

*** internal data**

The data that is described in a program and excludes all external data items and external file connectors. Items described in the LINKAGE SECTION of a program are treated as internal data.

*** internal data item**

A data item that is described in one program in a run unit. An internal data item can have a global name.

internal decimal data item

A data item that is described as USAGE PACKED-DECIMAL or USAGE COMP-3, and that has a PICTURE character string that defines the item as numeric (a valid combination of symbols 9, S, P, or V). Synonymous with *packed-decimal data item*.

*** internal file connector**

A file connector that is accessible to only one object program in the run unit.

internal floating-point data item

A data item that is described as USAGE COMP-1 or USAGE COMP-2. COMP-1 defines a single-precision floating-point data item. COMP-2 defines a double-precision floating-point data item. There is no PICTURE clause associated with an internal floating-point data item.

*** intrarecord data structure**

The entire collection of groups and elementary data items from a logical record that a contiguous subset of the data description entries defines. These data description entries include all entries

whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

intrinsic function

A predefined function, such as a commonly used arithmetic function, called by a built-in function reference.

*** invalid key condition**

A condition, at run time, caused when a specific value of the key associated with an indexed or relative file is determined to be not valid.

*** I-O-CONTROL**

The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

*** I-O-CONTROL entry**

An entry in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION; this entry contains clauses that provide information required for the transmission and handling of data on named files during the execution of a program.

*** I-O mode**

The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

*** I-O status**

A conceptual entity that contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

is-a

A relationship that characterizes classes and subclasses in an inheritance hierarchy. Subclasses that have an is-a relationship to a class inherit from that class.

ISPF

See *Interactive System Productivity Facility (ISPF)*.

iteration structure

A program processing logic in which a series of statements is repeated while a condition is true or until a condition is true.

J

J2EE

See *Java 2 Platform, Enterprise Edition (J2EE)*.

Java 2 Platform, Enterprise Edition (J2EE)

An environment for developing and deploying enterprise applications, defined by Oracle. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications. (Oracle)

Java Batch Launcher and Toolkit for z/OS (JZOS)

A set of tools that helps you develop z/OS Java applications that run in a traditional batch environment, and that access z/OS system services.

Java batch-processing program (JBP)

An IMS batch-processing program that has access to online databases and output message queues. JBPs run online, but like programs in a batch environment, they are started with JCL or in a TSO session.

Java batch-processing region

An IMS dependent region in which only Java batch-processing programs are scheduled.

Java Database Connectivity (JDBC)

A specification from Oracle that defines an API that enables Java programs to access databases.

Java message-processing program (JMP)

A Java application program that is driven by transactions and has access to online IMS databases and message queues.

Java message-processing region

An IMS dependent region in which only Java message-processing programs are scheduled.

Java Native Interface (JNI)

A programming interface that lets Java code that runs inside a Java virtual machine (JVM) interoperate with applications and libraries written in other programming languages.

Java virtual machine (JVM)

A software implementation of a central processing unit that runs compiled Java programs.

JavaBeans

A portable, platform-independent, reusable component model. (Oracle)

JBP

See *Java batch-processing program (JBP)*.

JDBC

See *Java Database Connectivity (JDBC)*.

JMP

See *Java message-processing program (JMP)*.

job control language (JCL)

A control language used to identify a job to an operating system and to describe the job's requirements.

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format.

JVM

See *Java virtual machine (JVM)*.

JZOS

See *Java Batch Launcher and Toolkit for z/OS*.

K**K**

When referring to storage capacity, two to the tenth power; 1024 in decimal notation.

*** key**

A data item that identifies the location of a record, or a set of data items that serve to identify the ordering of data.

*** key of reference**

The key, either prime or alternate, currently being used to access records within an indexed file.

*** keyword**

A context-sensitive word or a reserved word whose presence is required when the format in which the word appears is used in a source unit.

kilobyte (KB)

One kilobyte equals 1024 bytes.

L*** language-name**

A system-name that specifies a particular programming language.

Language Environment

Short form of z/OS Language Environment. A set of architectural constructs and interfaces that provides a common runtime environment and runtime services for C, C++, COBOL, FORTRAN and PL/I applications. It is required for programs compiled by Language Environment-conforming compilers and for Java applications.

Language Environment-conforming

A characteristic of compiler products (such as Enterprise COBOL, COBOL for OS/390 & VM, COBOL for MVS & VM, C/C++ for MVS & VM, PL/I for MVS & VM) that produce object code conforming to the Language Environment conventions.

last-used state

A state that a program is in if its internal values remain the same as when the program was exited (the values are not reset to their initial values).

*** letter**

A character belonging to one of the following two sets:

1. Uppercase letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
2. Lowercase letters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

*** level indicator**

Two alphabetic characters that identify a specific type of file or a position in a hierarchy. The level indicators in the DATA DIVISION are: CD, FD, and SD.

*** level-number**

A user-defined word (expressed as a two-digit number) that indicates the hierarchical position of a data item or the special properties of a data description entry. Level-numbers in the range from 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 can be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

*** library-name**

A user-defined word that names a COBOL library that the compiler is to use for compiling a given source program.

*** library text**

A sequence of text words, comment lines, inline comments, the separator space, or the separator pseudo-text delimiter in a COBOL library.

Lilian date

The number of days since the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582. The Lilian date format is named in honor of Luigi Lilio, the creator of the Gregorian calendar.

*** lineage-counter**

A special register whose value points to the current position within the page body.

link

(1) The combination of the link connection (the transmission medium) and two link stations, one at each end of the link connection. A link can be shared among multiple links in a multipoint or token-ring configuration. (2) To interconnect items of data or portions of one or more computer programs; for example, linking object programs by a linkage-editor to produce an executable file.

LINKAGE SECTION

The section in the DATA DIVISION of the called program or invoked method that describes data items available from the calling program or invoking method. Both the calling program or invoking method and the called program or invoked method can refer to these data items.

linker

A term that refers to either the z/OS binder (linkage-editor).

literal

A character string whose value is specified either by the ordered set of characters comprising the string or by the use of a figurative constant.

little-endian

The default format that Intel processors use to store binary data and UTF-16 characters. In this format, the most significant byte of a binary data item is at the highest address and the most significant byte of a UTF-16 character is at the highest address. Compare with *big-endian*.

local reference

A reference to an object that is within the scope of your method.

locale

A set of attributes for a program execution environment that indicates culturally sensitive considerations, such as character code page, collating sequence, date and time format, monetary value representation, numeric value representation, or language.

*** LOCAL-STORAGE SECTION**

The section of the DATA DIVISION that defines storage that is allocated and freed on a per-invocation basis, depending on the value assigned in the VALUE clauses.

*** logical operator**

One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND, or OR, or both can be used as logical connectives. NOT can be used for logical negation.

*** logical record**

The most inclusive data item. The level-number for a record is 01. A record can be either an elementary item or a group of items. Synonymous with *record*.

*** low-order end**

The rightmost character of a string of characters.

M

main program

In a hierarchy of programs and subroutines, the first program that receives control when the programs are run within a process.

makefile

A text file that contains a list of the files for your application. The make utility uses this file to update the target files with the latest changes.

*** mass storage**

A storage medium in which data can be organized and maintained in both a sequential manner and a nonsequential manner.

*** mass storage device**

A device that has a large storage capacity, such as a magnetic disk.

*** mass storage file**

A collection of records that is stored in a mass storage medium.

*** megabyte (MB)**

One megabyte equals 1,048,576 bytes.

*** merge file**

A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

message-processing program (MPP)

An IMS application program that is driven by transactions and has access to online IMS databases and message queues.

message queue

The data set on which messages are queued before being processed by an application program or sent to a terminal.

method

Procedural code that defines an operation supported by an object and that is executed by an INVOKE statement on that object.

*** method definition**

The COBOL source code that defines a method.

*** method identification entry**

An entry in the METHOD-ID paragraph of the IDENTIFICATION DIVISION; this entry contains a clause that specifies the method-name.

method invocation

A communication from one object to another that requests the receiving object to execute a method.

method-name

The name of an object-oriented operation. When used to invoke the method, the name can be an alphanumeric or national literal or a category alphanumeric or category national data item. When used in the METHOD-ID paragraph to define the method, the name must be an alphanumeric or national literal.

method hiding

See *hide*.

method overloading

See *overload*.

method overriding

See *override*.

*** mnemonic-name**

A user-defined word that is associated in the ENVIRONMENT DIVISION with a specified implementor-name.

module definition file

A file that describes the code segments within a program object.

MPP

See *message-processing program (MPP)*.

multitasking

A mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks.

multithreading

Concurrent operation of more than one path of execution within a computer. Synonymous with *multiprocessing*.

N**name**

A word (composed of not more than 30 characters) that defines a COBOL operand.

namespace

See *XML namespace*.

national character

(1) A UTF-16 character in a USAGE NATIONAL data item or national literal. (2) Any character represented in UTF-16.

national character data

A general reference to data represented in UTF-16.

national character position

See *character position*.

national data

See *national character data*.

national data item

A data item of category national, national-edited, or numeric-edited of USAGE NATIONAL.

national decimal data item

An external decimal data item that is described implicitly or explicitly as USAGE NATIONAL and that contains a valid combination of PICTURE symbols 9, S, P, and V.

national-edited data item

A data item that is described by a PICTURE character string that contains at least one instance of the symbol N and at least one of the simple insertion symbols B, 0, or /. A national-edited data item has USAGE NATIONAL.

national floating-point data item

An external floating-point data item that is described implicitly or explicitly as USAGE NATIONAL and that has a PICTURE character string that describes a floating-point data item.

national group item

A group item that is explicitly or implicitly described with a GROUP-USAGE NATIONAL clause. A national group item is processed as though it were defined as an elementary data item of category national for operations such as INSPECT, STRING, and UNSTRING. This processing ensures correct padding and truncation of national characters, as contrasted with defining USAGE NATIONAL data items within an alphanumeric group item. For operations that require processing of the elementary

items within a group, such as MOVE CORRESPONDING, ADD CORRESPONDING, and INITIALIZE, a national group is processed using group semantics.

*** native character set**

The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

*** native collating sequence**

The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

native method

A Java method with an implementation that is written in another programming language, such as COBOL.

*** negated combined condition**

The NOT logical operator immediately followed by a parenthesized combined condition. See also *condition* and *combined condition*.

*** negated simple condition**

The NOT logical operator immediately followed by a simple condition. See also *condition* and *simple condition*.

nested program

A program that is directly contained within another program.

*** next executable sentence**

The next sentence to which control will be transferred after execution of the current statement is complete.

*** next executable statement**

The next statement to which control will be transferred after execution of the current statement is complete.

*** next record**

The record that logically follows the current record of a file.

*** noncontiguous items**

Elementary data items in the WORKING-STORAGE SECTION and LINKAGE SECTION that bear no hierarchic relationship to other data items.

*** noncontiguous items**

Elementary data items in the WORKING-STORAGE and LINKAGE SECTIONS that bear no hierarchic relationship to other data items.

*** nonnumeric item**

A data item whose description permits its content to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

null

A figurative constant that is used to assign, to pointer data items, the value of an address that is not valid. NULLS can be used wherever NULL can be used.

*** numeric character**

A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

numeric data item

(1) A data item whose description restricts its content to a value represented by characters chosen from the digits 0 through 9. If signed, the item can also contain a +, -, or other representation of an operational sign. (2) A data item of category numeric, internal floating-point, or external floating-point. A numeric data item can have USAGE DISPLAY, NATIONAL, PACKED-DECIMAL, BINARY, COMP, COMP-1, COMP-2, COMP-3, COMP-4, or COMP-5.

numeric-edited data item

A data item that contains numeric data in a form suitable for use in printed output. The data item can consist of external decimal digits from 0 through 9, the decimal separator, commas, the currency sign,

sign control characters, and other editing characters. A numeric-edited item can be represented in either USAGE DISPLAY or USAGE NATIONAL.

*** numeric function**

A function whose class and category are numeric but that for some possible evaluation does not satisfy the requirements of integer functions.

*** numeric item**

A data item whose description restricts its content to a value represented by characters chosen from the digits from '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

*** numeric literal**

A literal composed of one or more numeric characters that can contain a decimal point or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

O

object

An entity that has state (its data values) and operations (its methods). An object is a way to encapsulate state and behavior. Each object in the class is said to be an instance of the class.

object code

Output from a compiler or assembler that is itself executable machine code or is suitable for processing to produce executable machine code.

*** OBJECT-COMPUTER**

The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, where the object program is run, is described.

*** object computer entry**

An entry in the OBJECT-COMPUTER paragraph of the ENVIRONMENT DIVISION; this entry contains clauses that describe the computer environment in which the object program is to be executed.

object deck

A portion of an object program suitable as input to a linkage-editor. Synonymous with *object module* and *text deck*.

object instance

A single object, of possibly many, instantiated from the specifications in the object paragraph of a COBOL class definition. An object instance has a copy of all the data described in its class definition and all inherited data. The methods associated with an object instance includes the methods defined in its class definition and all inherited methods.

An object instance can be an instance of a Java class.

object module

Synonym for *object deck* or *text deck*.

*** object of entry**

A set of operands and reserved words, within a DATA DIVISION entry of a COBOL program, that immediately follows the subject of the entry.

object-oriented programming

A programming approach based on the concepts of encapsulation and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates on the data objects that comprise the problem and how they are manipulated, not on how something is accomplished.

object program

A set or group of executable machine-language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program or class definition. Where there is no danger of ambiguity, the word *program* can be used in place of *object program*.

object reference

A value that identifies an instance of a class. If the class is not specified, the object reference is universal and can apply to instances of any class.

*** object time**

The time at which an object program is executed. Synonymous with *run time*.

*** obsolete element**

A COBOL language element in the 85 COBOL Standard that was deleted from the 2002 COBOL Standard.

ODO object

In the example below, X is the object of the OCCURS DEPENDING ON clause (ODO object).

```
WORKING-STORAGE SECTION.  
01  TABLE-1.  
    05  X          PIC S9.  
    05  Y OCCURS 3 TIMES  
        DEPENDING ON X  PIC X.
```

The value of the ODO object determines how many of the ODO subject appear in the table.

ODO subject

In the example above, Y is the subject of the OCCURS DEPENDING ON clause (ODO subject). The number of Y ODO subjects that appear in the table depends on the value of X.

*** open mode**

The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

*** operand**

(1) The general definition of operand is "the component that is operated upon." (2) For the purposes of this document, any lowercase word (or words) that appears in a statement or entry format can be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

operation

A service that can be requested of an object.

*** operational sign**

An algebraic sign that is associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

optional file

A file that is declared as being not necessarily available each time the object program is run.

*** optional word**

A reserved word that is included in a specific format only to improve the readability of the language. Its presence is optional to the user when the format in which the word appears is used in a source unit.

*** output file**

A file that is opened in either output mode or extend mode.

*** output mode**

The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

*** output procedure**

A set of statements to which control is given during execution of a format 1 SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

overflow condition

A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overload

To define a method with the same name as another method that is available in the same class, but with a different signature. See also *signature*.

override

To redefine an instance method (inherited from a parent class) in a subclass.

P**package**

A group of related Java classes, which can be imported individually or as a whole.

packed-decimal data item

See *internal decimal data item*.

padding character

An alphanumeric or national character that is used to fill the unused character positions in a physical record.

page

A vertical division of output data that represents a physical separation of the data. The separation is based on internal logical requirements or external characteristics of the output medium or both.

*** page body**

That part of the logical page in which lines can be written or spaced or both.

*** paragraph**

In the PROCEDURE DIVISION, a paragraph-name followed by a separator period and by zero, one, or more sentences. In the IDENTIFICATION DIVISION and ENVIRONMENT DIVISION, a paragraph header followed by zero, one, or more entries.

*** paragraph header**

A reserved word, followed by the separator period, that indicates the beginning of a paragraph in the IDENTIFICATION DIVISION and ENVIRONMENT DIVISION. The permissible paragraph headers in the IDENTIFICATION DIVISION are:

```
PROGRAM-ID. (Program IDENTIFICATION
             DIVISION)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION
            DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

The permissible paragraph headers in the ENVIRONMENT DIVISION are:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class
             CONFIGURATION SECTION)
FILE-CONTROL.
I-O-CONTROL.
```

*** paragraph-name**

A user-defined word that identifies and begins a paragraph in the PROCEDURE DIVISION.

parameter

(1) Data passed between a calling program and a called program. (2) A data element in the USING phrase of a method invocation. Arguments provide additional information that the invoked method can use to perform the requested operation.

Persistent Reusable JVM

A JVM that can be serially reused for transaction processing by resetting the JVM between transactions. The reset phase restores the JVM to a known initialization state.

*** phrase**

An ordered set of one or more consecutive COBOL character strings that form a portion of a COBOL procedural statement or of a COBOL clause.

*** physical record**

See *block*.

pointer data item

A data item in which address values can be stored. Data items are explicitly defined as pointers with the `USAGE IS POINTER` clause. `ADDRESS OF` special registers are implicitly defined as pointer data items. Pointer data items can be compared for equality or moved to other pointer data items.

port

(1) To modify a computer program to enable it to run on a different platform. (2) In the Internet suite of protocols, a specific logical connector between the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) and a higher-level protocol or application. A port is identified by a port number.

portability

The ability to transfer an application program from one application platform to another with relatively few changes to the source program.

precomposed character

A single Unicode character that can be represented using two or more Unicode characters through a canonical decomposition. A precomposed character does not have the same physical representation as its composed character form. For example, Unicode character U+00E4 (ä) is a precomposed character that can be represented as a combination of Unicode characters U+0061 + U+0308 (a - latin small letter a + combining diaeresis). A precomposed character is typically used to represent a latin letter with a diacritical mark or some other combining character.

preinitialization

The initialization of the COBOL runtime environment in preparation for multiple calls from programs, especially non-COBOL programs. The environment is not terminated until an explicit termination.

*** prime record key**

A key whose contents uniquely identify a record within an indexed file.

*** priority-number**

A user-defined word that classifies sections in the `PROCEDURE DIVISION` for purposes of segmentation. Segment numbers can contain only the characters 0 through 9. A segment number can be expressed as either one or two digits.

private

As applied to factory data or instance data, accessible only by methods of the class that defines the data.

*** procedure**

A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the `PROCEDURE DIVISION`.

*** procedure branching statement**

A statement that causes the explicit transfer of control to a statement other than the next executable statement in the sequence in which the statements are written in the source code. The procedure branching statements are: `ALTER`, `CALL`, `EXIT`, `EXIT PROGRAM`, `GO TO`, `MERGE` (with the `OUTPUT PROCEDURE` phrase), `PERFORM` and `SORT` (with the `INPUT PROCEDURE` or `OUTPUT PROCEDURE` phrase), `XML PARSE`.

PROCEDURE DIVISION

The COBOL division that contains instructions for solving a problem.

procedure integration

One of the functions of the COBOL optimizer is to simplify calls to performed procedures or contained programs.

`PERFORM` procedure integration is the process whereby a `PERFORM` statement is replaced by its performed procedures. Contained program procedure integration is the process where a call to a contained program is replaced by the program code.

*** procedure-name**

A user-defined word that is used to name a paragraph or section in the `PROCEDURE DIVISION`. It consists of a paragraph-name (which can be qualified) or a section-name.

procedure pointer

A data item in which a pointer to an entry point can be stored. A data item defined with the USAGE IS PROCEDURE-POINTER clause contains the address of a procedure entry point.

procedure-pointer data item

A data item in which a pointer to an entry point can be stored. A data item defined with the USAGE IS PROCEDURE-POINTER clause contains the address of a procedure entry point. Typically used to communicate with COBOL and Language Environment programs.

process

The course of events that occurs during the execution of all or part of a program. Multiple processes can run concurrently, and programs that run within a process can share resources.

program

(1) A sequence of instructions suitable for processing by a computer. Processing may include the use of a compiler to prepare the program for execution, as well as a runtime environment to execute it. (2) A logical assembly of one or more interrelated modules. Multiple copies of the same program can be run in different processes.

program-name

In the IDENTIFICATION DIVISION and the end program marker, a user-defined word or an alphanumeric literal that identifies a COBOL source program.

*** program identification entry**

In the PROGRAM-ID paragraph of the IDENTIFICATION DIVISION, an entry that contains clauses that specify the program-name and assign selected program attributes to the program.

program-name

In the IDENTIFICATION DIVISION and the end program marker, a user-defined word or alphanumeric literal that identifies a COBOL source program.

project

The complete set of data and actions that are required to build a target, such as a dynamic link library (DLL) or other executable (EXE).

*** pseudo-text**

A sequence of text words, comment lines, inline comments, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

*** pseudo-text delimiter**

Two contiguous equal sign characters (==) used to delimit pseudo-text.

*** punctuation character**

A character that belongs to the following set:

Character	Meaning
,	Comma
;	Semicolon
:	Colon
.	Period (full stop)
"	Quotation mark
(Left parenthesis
)	Right parenthesis
	Space
=	Equal sign

Q

QSAM (Queued Sequential Access Method)

An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

*** qualified data-name**

An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

*** qualifier**

(1) A data-name or a name associated with a level indicator that is used in a reference either together with another data-name (which is the name of an item that is subordinate to the qualifier) or together with a condition-name. (2) A section-name that is used in a reference together with a paragraph-name specified in that section. (3) A library-name that is used in a reference together with a text-name associated with that library.

R*** random access**

An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

*** record**

See *logical record*.

*** record area**

A storage area allocated for the purpose of processing the record described in a record description entry in the FILE SECTION of the DATA DIVISION. In the FILE SECTION, the current number of character positions in the record area is determined by the explicit or implicit RECORD clause.

*** record description**

See *record description entry*.

*** record description entry**

The total set of data description entries associated with a particular record. Synonymous with *record description*.

recording mode

The format of the logical records in a file. Recording mode can be F (fixed-length), V (variable-length), S (spanned), or U (undefined).

record key

A key whose contents identify a record within an indexed file.

*** record-name**

A user-defined word that names a record described in a record description entry in the DATA DIVISION of a COBOL program.

*** record number**

The ordinal number of a record in the file whose organization is sequential.

recording mode

The format of the logical records in a file. Recording mode can be F (fixed length), V (variable length), S (spanned), or U (undefined).

recursion

A program calling itself or being directly or indirectly called by one of its called programs.

recursively capable

A program is recursively capable (can be called recursively) if the RECURSIVE attribute is on the PROGRAM-ID statement.

reel

A discrete portion of a storage medium, the dimensions of which are determined by each implementor that contains part of a file, all of a file, or any number of files. Synonymous with *unit* and *volume*.

reentrant

The attribute of a program or routine that lets more than one user share a single copy of a program object.

*** reference format**

A format that provides a standard method for describing COBOL source programs.

reference modification

A method of defining a new category alphanumeric, category DBCS, or category national data item by specifying the leftmost character and length relative to the leftmost character position of a USAGE DISPLAY, DISPLAY-1, or NATIONAL data item.

*** reference-modifier**

A syntactically correct combination of character strings and separators that defines a unique data item. It includes a delimiting left parenthesis separator, the leftmost character position, a colon separator, optionally a length, and a delimiting right parenthesis separator.

*** relation**

See *relational operator* or *relation condition*.

*** relation character**

A character that belongs to the following set:

Character	Meaning
>	Greater than
<	Less than
=	Equal to

*** relation condition**

The proposition (for which a truth value can be determined) that the value of an arithmetic expression, data item, alphanumeric literal, or index-name has a specific relationship to the value of another arithmetic expression, data item, alphanumeric literal, or index name. See also *relational operator*.

*** relational operator**

A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

Character	Meaning
IS GREATER THAN	Greater than
IS >	Greater than
IS NOT GREATER THAN	Not greater than
IS NOT >	Not greater than
IS LESS THAN	Less than
IS <	Less than
IS NOT LESS THAN	Not less than
IS NOT <	Not less than
IS EQUAL TO	Equal to
IS =	Equal to
IS NOT EQUAL TO	Not equal to
IS NOT =	Not equal to
IS GREATER THAN OR EQUAL TO	Greater than or equal to
IS >=	Greater than or equal to

Character	Meaning
IS LESS THAN OR EQUAL TO	Less than or equal to
IS <=	Less than or equal to

*** relative file**

A file with relative organization.

*** relative key**

A key whose contents identify a logical record in a relative file.

*** relative organization**

The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the logical ordinal position of the record in the file.

*** relative record number**

The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal that is an integer.

*** reserved word**

A COBOL word that is specified in the list of words that can be used in a COBOL source program, but that must not appear in the program as a user-defined word or system-name.

*** resource**

A facility or service, controlled by the operating system, that an executing program can use.

*** resultant identifier**

A user-defined data item that is to contain the result of an arithmetic operation.

reusable environment

A reusable environment is created when you establish an assembler program as the main program by using either the old COBOL interfaces for preinitialization (RTEREUS runtime option), or the Language Environment interface, CEEPIPI.

routine

A set of statements in a COBOL program that causes the computer to perform an operation or series of related operations. In Language Environment, refers to either a procedure, function, or subroutine.

*** routine-name**

A user-defined word that identifies a procedure written in a language other than COBOL.

*** run time**

The time at which an object program is executed. Synonymous with *object time*.

runtime environment

The environment in which a COBOL program executes.

*** run unit**

A stand-alone object program, or several object programs, that interact by means of COBOL CALL or INVOKE statements and function at run time as an entity.

A run unit is also called an enclave in Language Environment terminology.

S

SBCS

See *single-byte character set (SBCS)*.

scope terminator

A COBOL reserved word that marks the end of certain PROCEDURE DIVISION statements. It can be either explicit (END-ADD, for example) or implicit (separator period).

*** section**

A set of zero, one, or more paragraphs or entities, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

*** section header**

A combination of words followed by a separator period that indicates the beginning of a section in any of these divisions: ENVIRONMENT, DATA, or PROCEDURE. In the ENVIRONMENT DIVISION and

DATA DIVISION, a section header is composed of reserved words followed by a separator period. The permissible section headers in the ENVIRONMENT DIVISION are:

```
CONFIGURATION SECTION.  
INPUT-OUTPUT SECTION.
```

The permissible section headers in the DATA DIVISION are:

```
FILE SECTION.  
WORKING-STORAGE SECTION.  
LOCAL-STORAGE SECTION.  
LINKAGE SECTION.
```

In the PROCEDURE DIVISION, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a separator period.

*** section-name**

A user-defined word that names a section in the PROCEDURE DIVISION.

segmentation

A feature of Enterprise COBOL that is based on the 85 COBOL Standard segmentation module. The segmentation feature uses priority-numbers in section headers to assign sections to fixed segments or independent segments. Segment classification affects whether procedures contained in a segment receive control in initial state or last-used state.

selection structure

A program processing logic in which one or another series of statements is executed, depending on whether a condition is true or false.

*** sentence**

A sequence of one or more statements, the last of which is terminated by a separator period.

*** separately compiled program**

A program that, together with its contained programs, is compiled separately from all other programs.

*** separator**

A character or two or more contiguous characters used to delimit character strings.

*** separator comma**

A comma (,) followed by a space used to delimit character strings.

*** separator period**

A period (.) followed by a space used to delimit character strings.

*** separator semicolon**

A semicolon (;) followed by a space used to delimit character strings.

sequence of programs

A sequence of separate COBOL programs in a single source file that can be input to the compiler.

A sequence of programs is also called a *batch compilation* or a *compilation group*.

sequence structure

A program processing logic in which a series of statements is executed in sequential order.

*** sequential access**

An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

*** sequential file**

A file with sequential organization.

*** sequential organization**

The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

serial search

A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last.

session bean

In EJB, an enterprise bean that is created by a client and that usually exists only for the duration of a single client/server session. (Oracle)

77-level-description-entry

A data description entry that describes a noncontiguous data item that has level-number 77.

*** sign condition**

The proposition (for which a truth value can be determined) that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

signature

(1) The name of an operation and its parameters. (2) The name of a method and the number and types of its formal parameters.

*** simple condition**

Any single condition chosen from this set:

- Relation condition
- Class condition
- Condition-name condition
- Switch-status condition
- Sign condition

See also *condition* and *negated simple condition*.

single-byte character set (SBCS)

A set of characters in which each character is represented by a single byte. See also *ASCII* and *EBCDIC (Extended Binary-Coded Decimal Interchange Code)*.

slack bytes (within records)

Bytes inserted by the compiler between data items to ensure correct alignment of some elementary data items. Slack bytes contain no meaningful data. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment.

slack bytes (between records)

Bytes inserted by the programmer between blocked logical records of a file, to ensure correct alignment of some elementary data items. In some cases, slack bytes between records improve performance for records processed in a buffer.

*** sort file**

A collection of records to be sorted by a format 1 SORT statement. The sort file is created and can be used by the sort function only.

*** sort-merge file description entry**

An entry in the FILE SECTION of the DATA DIVISION that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

*** SOURCE-COMPUTER**

The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, where the source program is compiled, is described.

*** source computer entry**

An entry in the SOURCE-COMPUTER paragraph of the ENVIRONMENT DIVISION; this entry contains clauses that describe the computer environment in which the source program is to be compiled.

*** source item**

An identifier designated by a SOURCE clause that provides the value of a printable item.

source program

Although a source program can be represented by other forms and symbols, in this document the term always refers to a syntactically correct set of COBOL statements. A COBOL source program

commences with the IDENTIFICATION DIVISION or a COPY statement and terminates with the end program marker, if specified, or with the absence of additional source program lines.

source unit

A unit of COBOL source code that can be separately compiled: a program or a class definition. Also known as a *compilation unit*.

special character

A character that belongs to the following set:

Character	Meaning
+	Plus sign
-	Minus sign (hyphen)
*	Asterisk
/	Slant (forward slash)
=	Equal sign
\$	Currency sign
,	Comma
;	Semicolon
.	Period (decimal point, full stop)
"	Quotation mark
'	Apostrophe
(Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than
:	Colon
_	Underscore

SPECIAL - NAMES

The name of an ENVIRONMENT DIVISION paragraph in which environment-names are related to user-specified mnemonic-names.

*** special names entry**

An entry in the SPECIAL - NAMES paragraph of the ENVIRONMENT DIVISION; this entry provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating implementor-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

*** special registers**

Certain compiler-generated storage areas whose primary use is to store information produced in conjunction with the use of a specific COBOL feature.

*** standard data format**

The concept used in describing the characteristics of data in a COBOL DATA DIVISION under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

*** statement**

A syntactically valid combination of words, literals, and separators, beginning with a verb, written in a COBOL source program.

structured programming

A technique for organizing and coding a computer program in which the program comprises a hierarchy of segments, each segment having a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the hierarchy.

*** subclass**

A class that inherits from another class. When two classes in an inheritance relationship are considered together, the subclass is the inheritor or inheriting class; the superclass is the inheritee or inherited class.

*** subject of entry**

An operand or reserved word that appears immediately following the level indicator or the level-number in a DATA DIVISION entry.

*** subprogram**

See *called program*.

*** subscript**

An occurrence number that is represented by either an integer, a data-name optionally followed by an integer with the operator + or -, or an index-name optionally followed by an integer with the operator + or -, that identifies a particular element in a table. A subscript can be the word ALL when the subscripted identifier is used as a function argument for a function allowing a variable number of arguments.

*** subscripted data-name**

An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

substitution character

A character that is used in a conversion from a source code page to a target code page to represent a character that is not defined in the target code page.

*** superclass**

A class that is inherited by another class. See also *subclass*.

surrogate pair

In the UTF-16 format of Unicode, a pair of encoding units that together represents a single Unicode graphic character. The first unit of the pair is called a *high surrogate* and the second a *low surrogate*. The code value of a high surrogate is in the range X'D800' through X'DBFF'. The code value of a low surrogate is in the range X'DC00' through X'DFFF'. Surrogate pairs provide for more characters than the 65,536 characters that fit in the Unicode 16-bit coded character set.

switch-status condition

The proposition (for which a truth value can be determined) that an UPSI switch, capable of being set to an on or off status, has been set to a specific status.

*** symbolic-character**

A user-defined word that specifies a user-defined figurative constant.

syntax

(1) The relationship among characters or groups of characters, independent of their meanings or the manner of their interpretation and use. (2) The structure of expressions in a language. (3) The rules governing the structure of a language. (4) The relationship among symbols. (5) The rules for the construction of a statement.

*** system-name**

A COBOL word that is used to communicate with the operating environment.

T*** table**

A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

*** table element**

A data item that belongs to the set of repeated items comprising a table.

text deck

Synonym for *object deck* or *object module*.

*** text-name**

A user-defined word that identifies library text.

*** text word**

A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or pseudo-text that is any of the following characters:

- A separator, except for space; a pseudo-text delimiter; and the opening and closing delimiters for alphanumeric literals. The right parenthesis and left parenthesis characters, regardless of context within the library, source program, or pseudo-text, are always considered text words.
- A literal including, in the case of alphanumeric literals, the opening quotation mark and the closing quotation mark that bound the literal.
- Any other sequence of contiguous COBOL characters except comment lines and the word COPY bounded by separators that are neither a separator nor a literal.

thread

A stream of computer instructions (initiated by an application within a process) that is in control of a process.

token

In the COBOL editor, a unit of meaning in a program. A token can contain data, a language keyword, an identifier, or other part of the language syntax.

top-down design

The design of a computer program using a hierarchic structure in which related functions are performed at each level of the structure.

top-down development

See *structured programming*.

trailer-label

(1) A data-set label that follows the data records on a unit of recording medium. (2) Synonym for *end-of-file label*.

troubleshoot

To detect, locate, and eliminate problems in using computer software.

*** truth value**

The representation of the result of the evaluation of a condition in terms of one of two values: true or false.

typed object reference

A data-name that can refer only to an object of a specified class or any of its subclasses.

U*** unary operator**

A plus (+) or a minus (-) sign that precedes a variable or a left parenthesis in an arithmetic expression and that has the effect of multiplying the expression by +1 or -1, respectively.

unbounded table

A table with OCCURS *integer-1* to UNBOUNDED instead of specifying *integer-2* as the upper bound.

Unicode

A universal character encoding standard that supports the interchange, processing, and display of text that is written in any of the languages of the modern world. There are multiple encoding schemes to represent Unicode, including UTF-8, UTF-16, and UTF-32. Enterprise COBOL supports Unicode using UTF-16 in big-endian format as the representation for the national data type.

Uniform Resource Identifier (URI)

A sequence of characters that uniquely names a resource; in Enterprise COBOL, the identifier of a namespace. URI syntax is defined by the document [*Uniform Resource Identifier \(URI\): Generic Syntax*](#).

unit

A module of direct access, the dimensions of which are determined by IBM.

universal object reference

A data-name that can refer to an object of any class.

unrestricted storage

In AMODE 31, unrestricted storage is below the 2 GB bar and can be above or below the 16 MB line.

In AMODE 64, unrestricted storage encompasses all the storage available to your program, both above and below the 2 GB bar.

*** unsuccessful execution**

The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but can affect status indicators.

UPSI switch

A program switch that performs the functions of a hardware switch. Eight are provided: UPSI-0 through UPSI-7.

URI

See *Uniform Resource Identifier (URI)*.

*** user-defined word**

A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

V*** variable**

A data item whose value can be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

variable-length item

A group item that contains a table described with the `DEPENDING` phrase of the `OCCURS` clause.

*** variable-length record**

A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

*** variable-occurrence data item**

A variable-occurrence data item is a table element that is repeated a variable number of times. Such an item must contain an `OCCURS DEPENDING ON` clause in its data description entry or be subordinate to such an item.

*** variably located group**

A group item following, and not subordinate to, a variable-length table in the same record. The group item can be an alphanumeric group or a national group.

*** variably located item**

A data item following, and not subordinate to, a variable-length table in the same record.

*** verb**

A word that expresses an action to be taken by a COBOL compiler or object program.

volume

A module of external storage. For tape devices it is a reel; for direct-access devices it is a unit.

volume switch procedures

System-specific procedures that are executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

VSAM file system

A file system that supports COBOL sequential, relative, and indexed organizations.

W**web service**

A modular application that performs specific tasks and is accessible through open protocols like HTTP and SOAP.

white space

Characters that introduce space into a document. They are:

- Space
- Horizontal tabulation
- Carriage return
- Line feed
- Next line

as named in the Unicode Standard.

*** word**

A character string of not more than 30 characters that forms a user-defined word, a system-name, a reserved word, or a function-name.

*** WORKING-STORAGE SECTION**

The section of the DATA DIVISION that describes WORKING-STORAGE data items, composed either of noncontiguous items or WORKING-STORAGE records or of both.

workstation

A generic term for computers, including personal computers, 3270 terminals, intelligent workstations, and UNIX terminals. Often a workstation is connected to a mainframe or to a network.

wrapper

An object that provides an interface between object-oriented code and procedure-oriented code. Using wrappers lets programs be reused and accessed by other systems.

X**x**

The symbol in a PICTURE clause that can hold any character in the character set of the computer.

XML

Extensible Markup Language. A standard metalanguage for defining markup languages that was derived from and is a subset of SGML. XML omits the more complex and less-used parts of SGML and makes it much easier to write applications to handle document types, author and manage structured information, and transmit and share structured information across diverse computing systems. The use of XML does not require the robust applications and processing that is necessary for SGML. XML is developed under the auspices of the World Wide Web Consortium (W3C).

XML data

Data that is organized into a hierarchical structure with XML elements. The data definitions are defined in XML element type declarations.

XML declaration

XML text that specifies characteristics of the XML document such as the version of XML being used and the encoding of the document.

XML document

A data object that is well formed as defined by the W3C XML specification.

XML namespace

A mechanism, defined by the W3C XML Namespace specifications, that limits the scope of a collection of element names and attribute names. A uniquely chosen XML namespace ensures the unique identity of an element name or attribute name across multiple XML documents or multiple contexts within an XML document.

XML schema

A mechanism, defined by the W3C, for describing and constraining the structure and content of XML documents. An XML schema, which is itself expressed in XML, effectively defines a class of XML documents of a given type, for example, purchase orders.

Z

z/OS UNIX file system

A collection of files and directories that are organized in a hierarchical structure and can be accessed by using z/OS UNIX.

zoned decimal data item

An external decimal data item that is described implicitly or explicitly as USAGE DISPLAY and that contains a valid combination of PICTURE symbols 9, S, P, and V. The content of a zoned decimal data item is represented in characters 0 through 9, optionally with a sign. If the PICTURE string specifies a sign and the SIGN IS SEPARATE clause is specified, the sign is represented as characters + or -. If SIGN IS SEPARATE is not specified, the sign is one hexadecimal digit that overlays the first 4 bits of the sign position (leading or trailing).

#

85 COBOL Standard

The COBOL language defined by the following standards:

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

2002 COBOL Standard

The COBOL language defined by the following standard:

- *INCITS/ISO/IEC 1989-2002, Information technology - Programming languages - COBOL*

2014 COBOL Standard

The COBOL language defined by the following standard:

- *INCITS/ISO/IEC 1989:2014, Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

List of resources

Enterprise COBOL for z/OS

COBOL for z/OS publications

You can find the following publications in the [Enterprise COBOL for z/OS library](#):

- *What's New*, SC31-5708-00
- *Customization Guide*, SC27-8712-03
- *Language Reference*, SC27-8713-03
- *Programming Guide*, SC27-8714-03
- *Migration Guide*, GC27-8715-03
- *Performance Tuning Guide*, SC27-9202-02
- *Messages and Codes*, SC27-4648-02
- *Program Directory*, GI13-4526-03
- *Licensed Program Specifications*, GI13-4532-03

Softcopy publications

The following collection kits contain Enterprise COBOL and other product publications. You can find them at <https://www.ibm.com/resources/publications>.

- *z/OS Software Products Collection*
- *z/OS and Software Products DVD Collection*

Support

If you have a problem using Enterprise COBOL for z/OS, see the following site that provides up-to-date support information: <https://www.ibm.com/support/pages/node/6560933>.

Related publications

z/OS library publications

You can find the following publications in the [z/OS library](#).

Run-Time Library Extensions

- *Common Debug Architecture Library Reference*
- *Common Debug Architecture User's Guide*
- *DWARF/ELF Extensions Library Reference*

z/Architecture

- *Principles of Operation*

z/OS DFSMS

- *Access Method Services for Catalogs*
- *Checkpoint/Restart*
- *Macro Instructions for Data Sets*
- *Using Data Sets*

- *Utilities*

z/OS DFSORT

- *Application Programming Guide*
- *Installation and Customization*

z/OS ISPF

- *Dialog Developer's Guide and Reference*
- *User's Guide Vol I*
- *User's Guide Vol II*

z/OS Language Environment

- *Concepts Guide*
- *Customization*
- *Debugging Guide*
- *Language Environment Vendor Interfaces*
- *Programming Guide*
- *Programming Reference*
- *Run-Time Messages*
- *Run-Time Application Migration Guide*
- *Writing Interlanguage Communication Applications*

z/OS MVS

- *JCL Reference*
- *JCL User's Guide*
- *Programming: Callable Services for High-Level Languages*
- *Program Management: User's Guide and Reference*
- *System Commands*
- *z/OS Unicode Services User's Guide and Reference*
- *z/OS XML System Services User's Guide and Reference*

z/OS TSO/E

- *Command Reference*
- *Primer*
- *User's Guide*

z/OS UNIX System Services

- *Command Reference*
- *Programming: Assembler Callable Services Reference*
- *User's Guide*

z/OS XL C/C++

- *Programming Guide*
- *Run-Time Library Reference*

CICS Transaction Server for z/OS

You can find the following publications in the [CICS library](#):

- *Developing CICS Applications*

- *API (EXEC CICS) Reference*
- *Developing CICS System Programs*
- *Global User Exit Reference*
- *XPI Reference*
- *Using EXCI with CICS*

COBOL Report Writer Precompiler

- *Programmer's Manual*, SC26-4301
- *Installation and Operation*, SC26-4302

Db2 for z/OS

You can find the following publications in the [Db2 library](#):

- *Application Programming and SQL Guide*
- *Command Reference*
- *SQL Reference*

IBM z/OS Debugger (formerly IBM Debug for z Systems and Debug Tool)

You can find information about IBM z/OS Debugger in the [IBM z/OS Debugger library](#).

IBM Developer for z/OS (formerly IBM Developer for z Systems)

You can find information about IBM Developer for z/OS in the [IBM Developer for z/OS library](#).

Note: IBM Developer for z/OS supersedes IBM Developer for z Systems and Rational® Developer for z Systems.

You can find the following publications by searching their publication numbers in the [IBM Publications Center](#).

IMS

- *Application Programming API Reference*, SC18-9699
- *Application Programming Guide*, SC18-9698

WebSphere® Application Server for z/OS

- *Applications*, SA22-7959

Softcopy publications for z/OS

The following collection kit contains z/OS and related product publications:

- *z/OS CD Collection Kit*, SK3T-4269

Java

- *IBM SDK for Java - Tools Documentation*, publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp
- *The Java 2 Enterprise Edition Developer's Guide*, download.oracle.com/javaee/1.2.1/devguide/html/DevGuideTOC.html
- *Java 2 on z/OS*, www.ibm.com/servers/eserver/zseries/software/java/
- *The Java EE 5 Tutorial*, download.oracle.com/javaee/5/tutorial/doc/
- *The Java Language Specification, Third Edition*, by Gosling et al., java.sun.com/docs/books/jls/

- *The Java Native Interface*, download.oracle.com/javase/1.5.0/docs/guide/jni/
- *JDK 5.0 Documentation*, download.oracle.com/javase/1.5.0/docs/

JSON

- JavaScript Object Notation (JSON), www.json.org

Unicode and character representation

- *Unicode*, www.unicode.org/
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

XML

- *Extensible Markup Language (XML)*, www.w3.org/XML/
- *Namespaces in XML 1.0*, www.w3.org/TR/xml-names/
- *Namespaces in XML 1.1*, www.w3.org/TR/xml-names11/
- *XML specification*, www.w3.org/TR/xml/

Index

A

- accessibility
 - keyboard navigation [87](#)
 - of Enterprise COBOL for z/OS [87](#)
 - of this information [87](#)
 - using z/OS [87](#)
- accessibility features for this product [87](#)
- ADATA compiler option [13](#)
- ADEXIT compiler option [13](#)
- ADV compiler option [14](#)
- AFP compiler option [14](#)
- ALOWCBL compiler option [15](#)
- ALOWCOPYLOC compiler option [15](#)
- ALOWDEFINE compiler option [16](#)
- ARCH compiler option [16](#)
- ARITH compiler option [18](#)
- assistive technologies [87](#)
- asterisk (*) for indicating fixed compiler options [13](#)
- AWO compiler option [18](#)

B

- Bibliography [137](#)
- BLOCK0 compiler option [19](#)
- BUF compiler option [19](#)

C

- CBL statement [15](#)
- CICS
 - coding programs to run under
 - SORT statement [8](#)
 - sorting under
 - change reserved-word table [8](#)
- CICS reserved word table [8](#)
- coding
 - programs to run under CICS
 - SORT statement [8](#)
- comment lines [99](#)
- comments
 - sending [xiii](#)
- COMPILE compiler option [21](#)
- compiler options
 - CONDCOMP [21](#)
 - conflicting options [11](#)
 - default values [1](#)
 - description of
 - ADATA [13](#)
 - ADEXIT [13](#)
 - ADV [14](#)
 - AFP [14](#)
 - ALOWCBL [15](#)
 - ALOWCOPYLOC [15](#)
 - ALOWDEFINE [16](#)

- compiler options (*continued*)
 - description of (*continued*)

- ARCH [16](#)
- ARITH [18](#)
- AWO [18](#)
- BLOCK0 [19](#)
- BUF [19](#)
- COMPILE [21](#)
- COPYRIGHT [22](#)
- CURRENCY [22](#)
- DATA [23](#)
- DBCS [24](#)
- DBCSXREF [24](#)
- DECK [25](#)
- DIAGTRUNC [25](#)
- DISPSIGN [26](#)
- DLL [27](#)
- DYNAM [27](#)
- EXPORT [28](#)
- FASTSRT [28](#)
- FLAG [29](#)
- FLAGSTD [30](#)
- HGPR [32](#)
- INEXIT [32](#)
- INITCHECK [33](#)
- INTDATE [35](#)
- INVDATA [35](#)
- LANGUAGE [38](#)
- LIBEXIT [39](#)
- LINECNT [40](#)
- LIST [40](#)
- LITCHAR [41](#)
- LP [41](#)
- MAP [42](#)
- MAXPCF [42](#)
- MDECK [43](#)
- MSGEXIT [44](#)
- NAME [44](#)
- NUM [45](#)
- NUMCHECK [45](#)
- NUMCLS [49](#)
- NUMPROC [50](#)
- OBJECT [50](#)
- OFFSET [51](#)
- OPTIMIZE [51](#)
- OUTDD [52](#)
- PARMCHECK [52](#)
- PGMNAME [53](#)
- PRTEXIT [54](#)
- QUALIFY [54](#)
- RENT [55](#)
- RMODE [56](#)
- RULES [57](#)
- SEQ [58](#)
- SERVICE [59](#)
- SMARTBIN [59](#)
- SOURCE [60](#)

compiler options (*continued*)
description of (*continued*)

- SPACE [60](#)
- SQL [61](#)
- SQLCCSID [61](#)
- SQLIMS [62](#)
- SSRANGE [63](#)
- STGOPT [64](#)
- SUPPRESS [64](#)
- TERM [65](#)
- TEST [65](#)
- THREAD [67](#)
- TRUNC [69](#)
- TUNE [70](#)
- VBREF [71](#)
- VLR [71](#)
- VSAMOPENFS [72](#)
- WORD [73](#)
- XMLPARSE [74](#)
- XREFOPT [74](#)
- ZONECHECK [75](#)
- ZONEDATA [75](#)
- ZWB [77](#)

fixed
 indicate with asterisk (*) [13](#)
 making compiler options fixed [7](#)

- INITIAL [34](#)
- INLINE [34](#)
- modifying [2](#), [79](#)
- planning worksheet [2](#)
- setting defaults for [1](#), [79](#)

CONDCOMP compiler option
 description [21](#)

COPYRIGHT compiler option [22](#)

CURRENCY compiler option [22](#)

customer support [137](#)

customization

- compiler options [11](#), [79](#)
- installation jobs
 - Enterprise COBOL [79](#)
- planning for [1](#)

D

DATA compiler option [23](#)

DBCS compiler option [24](#)

DBCSXREF compiler option [24](#)

DECK compiler option [25](#)

default reserved word table [8](#)

default values

- compiler options [1](#)

DIAGTRUNC compiler option [25](#)

disability [87](#)

DISPSIGN compiler option [26](#)

DLL compiler option [27](#)

DYNAM compiler option [27](#)

E

EGCS [108](#)

Enterprise COBOL

- disable [9](#)
- enable [9](#)

Enterprise COBOL (*continued*)

- job modification [79](#)

error messages

- flagging [29](#)

EXPORT compiler option [28](#)

F

FASTSRT option [28](#)

feedback

- sending [xiii](#)

fixed compiler options

- indicate with asterisk (*) [13](#)

- making compiler options fixed [7](#)

FLAG compiler option [29](#)

FLAGSTD compiler option [30](#)

floating comment indicators (*>) [110](#)

G

Glossary [93](#)

H

HGPR compiler option [32](#)

I

IGYCCICS (CICS reserved word table) [8](#)

IGYCDOPT

- link AMODE 31 and RMODE ANY [1](#)

- planning worksheet [2](#)

IGYCOPT

- syntax format [2](#)

IGYCRWT (default reserved word table) [8](#)

index checking [63](#)

INEXIT compiler option [32](#)

INITCHECK compiler option [33](#)

INITIAL compiler option

- description [34](#)

inline comments [112](#)

INLINE compiler option

- description [34](#)

installation option [15](#), [16](#)

INTDATE compiler option [35](#)

INVDATA compiler option [35](#)

K

keyboard navigation [87](#)

keyword [115](#)

L

LANGUAGE compiler option [38](#)

LIBEXIT compiler option [39](#)

LINECNT compiler option [40](#)

LIST compiler option [40](#)

List of resources [137](#)

LITCHAR compiler option [41](#)

LP compiler option [41](#)

M

macros
 IGYCDOPT (compiler options)
 planning worksheet [2](#)
 syntax format [2](#)
MAP compiler option [42](#)
MAXPCF compiler option [42](#)
MDECK compiler option [43](#)
messages, flagging [29](#)
MSGEXIT compiler option [44](#)

N

NAME compiler option [44](#)
nested programs [7](#)
nonoverridable compiler options, indicate with asterisk (*)
[13](#)
NUM compiler option [45](#)
NUMCHECK compiler option [45](#)
NUMCLS compiler option [49](#)
NUMPROC compiler option [50](#)

O

object code, reentrant [55](#)
OBJECT compiler option [50](#)
OFFSET compiler option [51](#)
OPTIMIZE compiler option [51](#)
optional words [xi](#)
OUTDD compiler option [52](#)

P

PARMCHECK compiler option [52](#)
PGMNAME compiler option [53](#)
planning worksheets
 description of [xii](#)
 IGYCDOPT (compiler options) [2](#)
preface [xi](#)
PROCESS (CBL) statement [15](#)
product registration [9](#)
product support [137](#)
PRTEXIT compiler option [54](#)
publications [137](#)

Q

QUALIFY compiler option [54](#)

R

RCFs
 sending [xiii](#)
reader comments
 sending [xiii](#)
reentrant object code [55](#)
RENT compiler option [55](#)
required words [xi](#)
reserved word table
 contents of [8](#)
 creating or modifying [81](#)
 nested programs [7](#)

reserved word table (*continued*)
 planning for [7](#)
 specifying an alternative table [73](#)
 supplied with Enterprise COBOL
 IGYCCICS (CICS) [8](#)
 IGYCRWT (default) [8](#)
residency mode [56](#)
RMODE compiler option [56](#)
RULES compiler option [57](#)

S

sample installation jobs [5](#)
sequence checking of line numbers [58](#)
SEQUENCE compiler option [58](#)
SERVICE compiler option [59](#)
shared storage
 placing Enterprise COBOL modules in [86](#)
SMARTBIN compiler option [59](#)
SORT statement
 under CICS
 change reserved-word table
 [8](#)
SOURCE compiler option [60](#)
SPACE compiler option [60](#)
SQL compiler option [61](#)
SQLCCSID compiler option [61](#)
SQLIMS compiler option [62](#)
SSRANGE compiler option [63](#)
stacked words [xi](#)
STGOPT compiler option [64](#)
subscript checking [63](#)
support [137](#)
SUPPRESS compiler option [64](#)
syntax checking [21](#)
syntax diagrams, how to read [xi](#)
syntax notation
 asterisk (*) [13](#)
 COBOL keywords [xii](#)
 repeat arrows [xii](#)
SYSLIN [50](#)
SYSOUT [52](#)
SYSPUNCH [25](#)
SYSTEM [65](#)

T

TERM compiler option [65](#)
TEST compiler option [65](#)
THREAD compiler option [67](#)
TRUNC compiler option [69](#)
TUNE compiler option [70](#)

U

user exit routine
 ADEXIT compiler option [13](#)
 INEXIT compiler option [32](#)
 LIBEXIT compiler option [39](#)
 MSGEXIT compiler option [44](#)
 PRTEXIT compiler option [54](#)

V

VBREF compiler option [71](#)

VLR compiler option [71](#)

VSAMOPENFS compiler option [72](#)

W

WORD compiler option [73](#)

X

XMLPARSE compiler option [74](#)

XREF compiler option [74](#)

XREFOPT option [74](#)

Z

ZONECHECK compiler option [75](#)

ZONEDATA compiler option [75](#)

ZWB compiler option [77](#)



Product Number: 5655-EC6

SC27-8712-03

