

z/OS



Language Environment Customization

Version 2 Release 1

Note

Before using this information and the product it supports, read the information in "Notices" on page 279.

This edition applies to Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 1991, 2013.**

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

Contents

Figures vii

Tables ix

About this document xi

Who should read this information xi

How to read syntax diagrams xi

 Symbols. xi

 Syntax items xi

 Syntax examples xii

z/OS information xiii

How to send your comments to IBM . . . xv

If you have a technical problem xv

z/OS Version 2 Release 1 summary of changes xvii

Summary of changes for Language Environment . . . xvii

Part 1. Language Environment Customization: General information . 1

Chapter 1. Customization overview . . . 3

Deciding whether and what to customize. 3

Chapter 2. Description of Language Environment target libraries 5

Chapter 3. Choosing your Language Environment runtime library access . . . 9

LNKLST. 9

STEPLIB. 9

Chapter 4. Placing Language Environment modules in link pack and LIBPACK 11

Tailoring the Fortran LIBPACKS 12

 Choices to make now 12

Listing the contents of Fortran LIBPACKS 13

 Modifying the JCL for AFHWLIST. 13

Deleting routines from Fortran LIBPACKS 14

 Steps for modifying the JCL to delete routines

 from a Fortran LIBPACK 14

Adding routines to Fortran LIBPACKS 15

 Steps for modifying the JCL for adding routines

 to a Fortran LIBPACK 15

Where to place the tailored Fortran LIBPACKS. . . . 16

Part 2. Language Environment Customization: Runtime options, exits, and procedures 17

Chapter 5. Customizing Language Environment runtime options 19

Creating system-level runtime option defaults with

CEEPRMxx 23

 CEEPRMxx parmlib member 24

 CEE= statement at IPL. 29

 SET CEE command. 30

 SETCEE command 31

 D CEE command 34

 CEEPRMCC - syntax checking under z/OS batch 36

 CEEPRMCK - syntax checking under TSO/E . . 38

Creating region-level runtime option defaults with

CEEOPT 39

 Sample invocation of CEEOPT within the

 CEERDOPT member 40

 Sample invocation of CEEOPT within the

 CEERCOPT member 41

 Sample invocation of CEEOPT within the

 CELQRDOP member 42

 CEEOPT invocation for CEEROPT (AMODE 31) 43

 CEEOPT invocation for CELQROPT (AMODE

 64) 43

 CEEOPT coding guidelines for CEEROPT and

 CELQROPT 43

Chapter 6. Language Environment runtime options 45

Cobol compatibility. 45

Runtime options. 45

ABPERC 46

ABTERMENC 47

AIXBLD (COBOL only) 49

ALL31 50

ANYHEAP 52

AUTOTASK | NOAUTOTASK (Fortran only) . . . 54

BELOWHEAP 54

CBLOPTS (COBOL only) 56

CBLPSHPOP (COBOL only) 57

CBLQDA (COBOL Only) 58

CEEDUMP 58

CHECK (COBOL only) 61

COUNTRY 61

DEBUG (COBOL only) 63

DEPTHCONDLMT 63

DYNDUMP 65

ENVAR. 68

ERRCOUNT 70

ERRUNIT (Fortran only) 71

FILEHIST (Fortran only) 72

FILETAG (C/C++ only) 72

HEAP 74

HEAP64 (AMODE 64 only) 77

HEAPCHK 79

HEAPOOLS (C/C++ and Enterprise PL/I only). . 81

HEAPOOLS64 (AMODE 64 only) 84

INFOMSGFILTER	86
INQPCOPN (Fortran only)	88
INTERRUPT	88
IOHEAP64 (AMODE 64 only)	89
LIBHEAP64 (AMODE 64 only)	91
LIBSTACK	92
MSGFILE	94
MSGQ	97
NATLANG	98
OCSTATUS (Fortran only)	100
PC (Fortran only)	101
PLITASKCOUNT (PL/I only)	102
POSIX	103
PROFILE	104
PRTUNIT (Fortran only)	105
PUNUNIT (Fortran only)	105
RDRUNIT (Fortran only)	106
RECPAD (Fortran only)	106
RPTOPTS	107
RPTSTG	108
RTEREUS (COBOL only)	110
SIMVRD (COBOL only)	112
STACK	113
STACK64 (AMODE 64 only)	116
STORAGE	118
TERMTHDACT	121
TEST NOTEST	127
THREADHEAP	129
THREADSTACK	131
THREADSTACK64 (AMODE 64 only)	134
TRACE	135
TRAP	137
UPSI (COBOL only)	140
USRHDLR	141
VCTRSAVE	142
XUFLOW	143

Chapter 7. Customizing user exits . . . 145

Example	146
Changing the assembler language user exit	146
Changing the installation-wide assembler language user exit (non-CICS)	147
Changing the installation-wide assembler language user exit (CICS)	147
Creating an application-specific assembler language user exit	148
Changing the high-level language user exit	148
Steps for modifying the JCL for CEEWHLLX	149
Customizing Language Environment abnormal termination exits	149
Creating a Language Environment abnormal termination exit	149
CEEEXTAN abnormal termination exit CSECT	150
Identifying the abnormal termination exit (non-CICS)	151
Identifying the abnormal termination exit (CICS)	152
Identifying the abnormal termination exit (AMODE 64)	152
Creating global user exit XPCFTCH (CICS)	153
Using XPCFTCH for an Enterprise PL/I routine	153
Using XPCFTCH for a PL/I routine	153

Using XPCFTCH for a C/C++ routine	153
Creating a load notification user exit	154
Identifying the load notification user exit	154
CEEBLNUE CSECT	154
CEEBLNUE sample	155
Creating a storage tuning user exit	156

Chapter 8. Customizing the cataloged procedures 157

Making the cataloged procedure library available to your jobs	157
Tailoring the cataloged procedures and CLISTs to your site	159

Chapter 9. Using Language Environment under CICS 161

Add program resource definitions for CICS	161
Add destination control table (DCT) entries	163
Specifying the side file interface to be used	165
Add Language Environment-CICS data sets to the CICS startup job stream	166
Language Environment automatic storage tuning for CICS	167
Enclaves eligible for automatic storage tuning	167
Automatic storage tuning behavior	167
Altering the automatic storage tuning behavior	168

Chapter 10. Using Language Environment under IMS 169

Initializing library routine retention	169
Ending library routine retention	169

Chapter 11. Customizing language-specific features. 171

Choices to make now.	171
Modifying the OS/VS COBOL compatibility library routines	171
OS/VS COBOL considerations.	172
Modifying the COBOL parameter list exit	173
Steps for modifying the JCL for IGZWAPXS	174
Modifying the COBOL runtime environment	174
Modifying COBOL reusable environment behavior	174
Modifying nested enclave behavior	175
Modifying COBOL formatted dump behavior	175
Modifying the behavior of the COBOL runtime environment.	176
Modifying the JCL for IGZWARRE	176
Modifying the COBOL debug file name	176
Using a COBOL debug file user exit.	177
Using the COBOL debug file user exit interface	177
COBOL debug file user exit samples	179
Changing the C/C++ locale time information	179
Modifying the JCL for EDCLLOCL	179

Part 3. Appendixes 181

Appendix A. Language Environment user exits 183

Assembler and HLL user exits.	183
When assembler and HLL user exits are invoked	183
CEEEXITA behavior during enclave initialization.	185
CEEEXITA behavior during enclave termination	185
CEEEXITA behavior during process termination	185
Specifying abend codes to be percolated by Language Environment	186
Actions taken for errors that occur within the exit.	186
CEEEXITA assembler user exit interface	186
Parameter values in the assembler user exit	190
Abnormal termination exit	193
Usage notes for AMODE 31 applications	193
Usage notes for AMODE 64 applications	194
Load notification user exit	195
Storage tuning user exit	198
Region initialization	199
Region termination	199
Enclave initialization	199
Enclave termination	199
New load module (CICS only)	200
Using the storage tuning user exit	200
Using the storage tuning user exit to collect information	201
Using the storage tuning user exit to provide storage values	201
Using the storage tuning user exit to provide storage values	202
Storage tuning user exit interface.	202

Appendix B. Using Fortran with Language Environment 217

Customizing for Fortran applications link-edited with Language Environment	217
Changing the unit attribute table default values	217
Customizing for Fortran applications link-edited with VS FORTRAN	223
Changing the unit attribute table default values	223

Changing VS FORTRAN runtime option defaults	229
Changing the error option table defaults	234
Customizing Fortran LIBPACKs	238
Contents of the Fortran LIBPACK AFHPRNAG	238
Contents of the Fortran LIBPACK AFHPRNBG	243
Contents of the Fortran LIBPACK AFH5RENA	243
Contents of the Fortran LIBPACK AFH5RENB	245

Appendix C. Modules eligible for the link pack area 247

Language Environment base modules	247
Language Environment C/C++ component modules	248
Language Environment COBOL component modules	249
Language Environment Fortran component modules	250
Language Environment PL/I component modules	264

Appendix D. National language support 271

Modifying the JCL for Japanese national language support	271
National language support country codes for Language Environment	271

Appendix E. Accessibility 275

Accessibility features	275
Using assistive technologies	275
Keyboard navigation of the user interface	275
Dotted decimal syntax diagrams	275

Notices 279

Policy for unsupported hardware.	280
Minimum supported hardware	281
Programming interface information	281
Trademarks	281

Index 283

Figures

1. Effect of DEPTHCONDLMT(3) on condition handling	64	10. CEESTX control block map	203
2. Default CEEEXTAN	151	11. CEESTX CICS-specific control block map	205
3. Updated CEEEXTAN	151	12. Mapping of the CEESTX storage values control block.	208
4. Sample of CEEBLNUE load notification user exit CSECT	155	13. CEESTX storage used control block map	212
5. Format of an output transient data queue	163	14. CEESTX storage allocated control block map	213
6. Example of DFHDCT macro	165	15. IBM-supplied macro instructions	221
7. Location of user exits	184	16. Modified IBM-supplied macro instructions	222
8. Interface for CEEBXITA assembler user exit	187	17. IBM-supplied macro instructions	227
9. CEELNUE control block map	196	18. Modified IBM-supplied macro instructions	228
		19. Modified IBM-supplied macro instructions	229

Tables

1. Syntax examples	xii	20. Register conventions for the COBOL debug file user exit	178
2. Description of data set target libraries for Language Environment	5	21. Parameter values in the assembler user exit (Part 1)	191
3. Language Environment sample IEALPAnn or PROGxx members in CEE.SCEESAMP	11	22. Parameter values in the assembler user exit (Part 2)	192
4. Making the trade-off: Performance time versus storage use.	12	23. Fortran LIBPACKs	238
5. SMP/E sample jobs for deleting routines from Fortran LIBPACKs	14	24. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNAG	238
6. SMP/E sample jobs for adding routines to Fortran LIBPACKs	15	25. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNBG	243
7. Runtime options, defaults, and recommendations for Language Environment	19	26. Routines eligible for inclusion in the Fortran LIBPACK AFH5RENA	243
8. Samples for creating region-level runtime option load modules	40	27. routines eligible for inclusion in the Fortran LIBPACK AFH5RENB.	246
9. Condition handling of 0Cx ABENDS in a CICS environment	124	28. Language Environment modules eligible for inclusion in the link pack area and the extended link pack area	247
10. Handling of software-raised conditions in a CICS environment	125	29. C/C++ modules eligible for inclusion in the link pack area and the extended link pack area.	248
11. TRAP runtime option settings	137	30. COBOL modules eligible for inclusion in the link pack area and the extended link pack area.	249
12. Sample customization jobs for the user exits	145	31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area.	251
13. Sample assembler user exits for Language Environment.	146	32. PL/I modules eligible for inclusion in the link pack area and the extended link pack area.	264
14. Language Environment invocation procedures in CEE.SCEEPROC.	157	33. JCL modifications for Japanese national language support	271
15. Deciding how to make cataloged procedures available to your jobs	158	34. Country codes	271
16. Cataloged procedures and CLISTs information	159		
17. Excluding programming language support under CICS	162		
18. Customizing programming languages with sample customization jobs	171		
19. Using the usermods in the IGZWZAP job to modify the COBOL compatibility library	172		

About this document

This document is designed to help you to customize IBM® z/OS® Language Environment®. The first usage of every term listed in the glossary is indicated by *italics*. You can find the definitions for these terms in *z/OS Language Environment Concepts Guide*.

Who should read this information

This information is intended for systems programmers and system administrators who plan to customize Language Environment. To use this information, you need to be familiar with z/OS, the publications that describe your system, and job control language (JCL).

How to read syntax diagrams

This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that may be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.

Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

For users accessing the Information Center using a screen reader, syntax diagrams are provided in dotted decimal format.

Symbols

The following symbols may be displayed in syntax diagrams:

Symbol

Definition

- ▶— Indicates the beginning of the syntax diagram.
- ▶ Indicates that the syntax diagram is continued to the next line.
- ▶— Indicates that the syntax is continued from the previous line.
- ▶ Indicates the end of the syntax diagram.

Syntax items

Syntax diagrams contain many different items. Syntax items include:

- Keywords - a command name or any other literal information.
- Variables - variables are italicized, appear in lowercase, and represent the name of values you can supply.
- Delimiters - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
- Operators - operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operations that may need to be performed.
- Fragment references - a part of a syntax diagram, separated from the diagram to show greater detail.

- Separators - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

Note: If a syntax diagram shows a character that is not alphanumeric (for example, parentheses, periods, commas, equal signs, a blank space), enter the character as part of the syntax.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

Item type

Definition

Required

Required items are displayed on the main path of the horizontal line.

Optional

Optional items are displayed below the main path of the horizontal line.

Default

Default items are displayed above the main path of the horizontal line.

Syntax examples

The following table provides syntax examples.

Table 1. Syntax examples


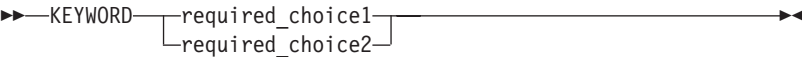
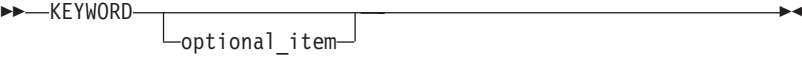
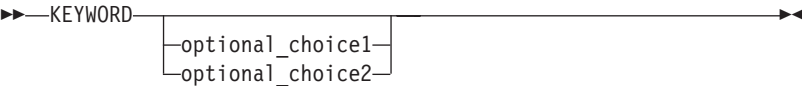
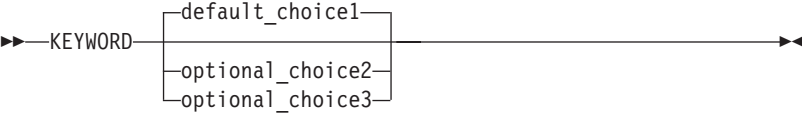

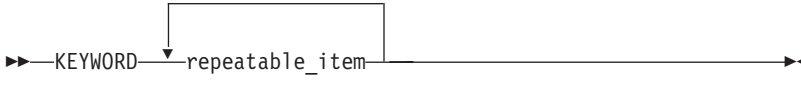
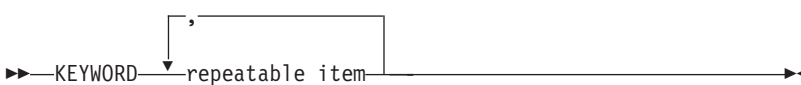
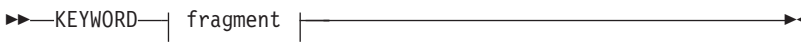
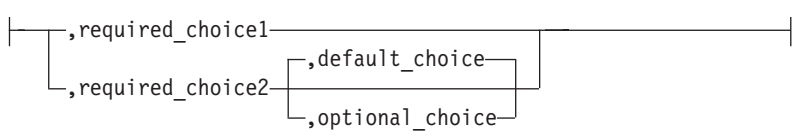
Item	Syntax example
Required item.	
Required items appear on the main path of the horizontal line. You must specify these items.	
Required choice.	
A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.	
Optional item.	
Optional items appear below the main path of the horizontal line.	
Optional choice.	
An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.	
Default.	
Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.	
Variable.	
Variables appear in lowercase italics. They represent names or values.	

Table 1. Syntax examples (continued)

Item	Syntax example
Repeatable item.	
An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.	
A character within the arrow means you must separate repeated items with that character.	
An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.	
Fragment.	
The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.	 <p data-bbox="657 661 1453 703">fragment:</p> 

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, including the z/OS Information Center, go to the z/OS Internet library (<http://www.ibm.com/systems/z/os/zos/bkserv/>).

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (<http://www.ibm.com/systems/z/os/zos/webqs.html>).
3. Mail the comments to the following address:
IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
US
4. Fax the comments to us, as follows:
From the United States and Canada: 1+845+432-9405
From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
z/OS V2R1.0 Language Environment Customization
SA38-0685-01
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (<http://www.ibm.com/systems/z/support/>).

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Summary of changes for Language Environment

The following changes are made to z/OS V2R1 as updated December 2013.

Changed

Information about COBOL V5.1 (or later) programs has been added to “Add Language Environment-CICS data sets to the CICS startup job stream” on page 166.

Part 1. Language Environment Customization: General information

This topic provides a Language Environment customization overview, description of Language Environment target libraries, how to choose your Language Environment runtime library access, and how to place Language Environment modules in Link Pack and LIBPACK.

Chapter 1. Customization overview

You can customize Language Environment by either tailoring and installing IBM-supplied usermods, or by tailoring and running specific jobs.

To tailor and install usermods:

1. Get the list of usermods that suit the programmer needs at your site. This topic will help you create this list.
2. Copy the customization jobs from the SCEESAMP data set into one of your private data sets so you will have unmodified copies of the jobs for your later reference and use.
3. Apply the usermods to the target libraries, **but do not accept them, and do not modify the distribution libraries.**
4. Use SMP/E RESTORE to remove a usermod if necessary (for example, if programming needs at your site change) or before you apply service to the modules it changes.
5. Reapply the usermod after successful installation of service.

To modify the JCL for customization jobs:

1. Copy the customization jobs from the SCEESAMP data set into one of your private data sets so you will have unmodified copies of the jobs for your later reference and use.
2. Add a job card appropriate for your site.
3. Add a JES Route card if your site requires one.
4. Modify the job according to the comments in the JCL or the instructions in this information.
5. Save and submit the job.
6. Most jobs will run with a condition code of 0. Check the description of each job to find out what condition code to expect. If the job did not run with the condition code you expected:
 - Check for an error message on the system console or the list output to find the cause of the problem.
 - Correct the problem.
 - Rerun the job.
 - Recheck the condition code.

Deciding whether and what to customize

Consider whether the IBM-supplied values for the runtime options provided with Language Environment suits the needs of your site. These values control such features as:

- The national language in which messages appear
- How a debug tool is invoked
- When condition handling is invoked
- How storage is allocated to the heap and stack
- How much storage is allocated
- The format of the program invocation character parameter
- Creation of a storage and runtime options report
- Shared storage allocations

|
| If you do not want to customize Language Environment now, you can put it into
| production using the IBM-supplied defaults. Or, you can use the instructions in
| this information to customize Language Environment later, if you choose. For
| many of the runtime options, application programmers can override the defaults in
| their code.

Application programmers at your site will be the primary users of Language Environment. Ask them what defaults they prefer for runtime options and user exits, which affect their work directly. Doing so ensures that the modifications you make will best support the application programs being developed at your site.

You need to make decisions about customizing:

- Runtime library access method (see Chapter 3, “Choosing your Language Environment runtime library access,” on page 9)
- Runtime options (see Chapter 5, “Customizing Language Environment runtime options,” on page 19)
- Assembler user exits (see Chapter 7, “Customizing user exits,” on page 145)
- Cataloged procedures (see Chapter 8, “Customizing the cataloged procedures,” on page 157)

You also need to decide:

- Whether to install some routines in the link pack area (see Chapter 4, “Placing Language Environment modules in link pack and LIBPACK,” on page 11)
- Whether to make Language Environment available under CICS® (see Chapter 9, “Using Language Environment under CICS,” on page 161)
- Whether to customize any programming language-specific features (see “Automatic storage tuning behavior” on page 167)

Chapter 2. Description of Language Environment target libraries

Table 2 provides a description of the Language Environment target libraries and when they are used. In most cases, the DDDEF entry for the data set is the same as the low-level qualifier. For the cases where this is not true, the appropriate DDDEF entry is listed. The high-level qualifier of these data sets may differ from customer to customer, but the default value is CEE.

The data sets in Table 2 have a legend associated with them in the rightmost columns of the table. The following descriptions explain the use and placing of these data sets.

- **AD** (Application Development) — These data sets are used during the assembly, compilation, or link-edit phases of application development. This does not include the procedures and CLISTs that can be used by application developers.
- **Ex** (Execution) — These data sets are used during execution of an application and must be placed in the program search order or be accessed directly through DD statements.
- **O** (Other) — These data sets contain sample jobs, source code, procedures, or CLISTs that are not used when assembling, compiling, link-editing, or executing programs.

Table 2. Description of data set target libraries for Language Environment

DDDEF entry	Data set name	Description	AD	Ex	O
	SAFHFORT	The Fortran-specific link-edit library that is used to resolve certain Fortran intrinsic function names. In link-edit steps, this library must precede SCEELKED if Fortran functions are needed.	X		
	SCEEBIND	Contains all Language Environment resident routines for XPLINK applications. This one library replaces the four libraries of resident routines for non-XPLINK applications (SCEELKED, SCEELKEX, SCEEOBJ, and SCEECPP). It must be used only when link-editing a program that includes XPLINK-compiled object modules. This data set will be eliminated in the near future and is being replaced with SCEEBND2. Customers should use the SCEEBND2 data set instead of SCEEBIND during XPLINK application development.	X		
	SCEEBND2	Contains all Language Environment resident routines for XPLINK applications. This one library replaces the four libraries of resident routines for non-XPLINK applications (SCEELKED, SCEELKEX, SCEEOBJ, and SCEECPP). It must be used only when link-editing a program that includes XPLINK-compiled object modules. The only difference between this data set and SCEEBIND is the record format. SCEEBND2 has a fixed blocked record format.	X		
	SCEECICS	Contains the COBOL-specific CICS runtime modules. Will only be used in the DHFRPL DD concatenation.		X	
	SCEECLST	Provides TSO/E CLISTs that C/C++ application developers can use.			X

Table 2. Description of data set target libraries for Language Environment (continued)

DDDEF entry	Data set name	Description	AD	Ex	O
	SCEECMAP	Contains the source for charmap files.			X
	SCEECPP	Contains Language Environment resident definitions that non-XPLINK C++ programs might need. This data set must be used whenever link-editing a non-XPLINK program that includes any C++ object.	X		
	SCEEGXLT	Contains the GENXLT source for the code set converters.			X
SCEEH	SCEEH	Contains ANSI C++ language headers used when compiling C++ programs.	X		
SCEEHARP	SCEEH.ARPA.H	Contains C-language headers used when compiling C programs.	X		
SCEEHH	SCEEH.H	Contains C-language headers used when compiling C programs.	X		
SCEEHNET	SCEEH.NET.H	Contains C-language headers used when compiling C programs.	X		
SCEEHNEI	SCEEH.NETINET.H	Contains C-language headers used when compiling C programs.	X		
SCEEHSYS	SCEEH.SYS.H	Contains C-language headers used when compiling C programs.	X		
SCEEHT	SCEEH.T	Contains ANSI C++ template files used when compiling C++ programs.	X		
	SCEELIB	Contains side-decks for DLLs provided by Language Environment. Many of the language-specific callable services available to XPLINK-compiled applications appear externally as DLL functions. To resolve these references from XPLINK applications, definition side-decks are required.	X		
	SCEELKED	Contains the link-edit stubs for non-XPLINK C/C++, PL/I, COBOL and Fortran languages and Language Environment-provided routines.	X		
	SCEELKEX	Contains non-XPLINK C/C++ stubs that are not in uppercase, truncated or mapped to another symbol. In link-edit steps this library must precede SCEELKED if unmapped names are used.	X		
	SCEELOCL	Provides the locale source files (pre-XP4).			X
	SCEELOCX	Provides the locale source files as defined by the XP4 standard.			X
	SCEELPA	Contains a subset of the SCEERUN modules that are reentrant and reside above the 16-MB line. This data set should be added to LPA1STxx for performance benefits.		X	
	SCEEMAC	Provides assembler macros to be used when writing assembler language code and using Language Environment services.	X		
	SCEEMSGP	Contains the message file to be used by the C pre-linker.	X		
	SCEEOBJ	Contains Language Environment resident definitions which may be required for non-XPLINK OS/390® UNIX System Services programs. This data set must be used whenever link-editing a non-XPLINK UNIX program.	X		
	SCEEPROC	Provides procedures used to link-edit and run Language Environment-conforming applications.			X

Table 2. Description of data set target libraries for Language Environment (continued)

DDDEF entry	Data set name	Description	AD	Ex	O
	SCEERUN	Contains the runtime library routines needed during execution of applications written in C/C++, PL/I, COBOL and FORTRAN.		X	
	SCEERUN2	Contains the runtime library routines needed during execution of applications, and those that require to reside in a PDSE.		X	
	SCEESAMP	Provides sample jobs, usermods, parmlib samples, some C headers, and some assembler macros.	X		
	SCEESPC	Provides the System Programmer C (SPC) routines to build free standing C applications. In link-edit steps, this library must precede SCEELKED in the SYSLIB DD concatenation.	X		
	SCEESPCO	Provides the object decks for the SPC routines for the SCEESPC data set.			X
	SIBMCALL	Provides the support for OS PL/I PLICALLA and PLICALLB entry points. In link-edit steps, this library must precede SCEELKED if PL/I for MVS™ and VM applications use OS PL/I PLICALLA or PLICALLB as entry points.	X		
	SIBMCAL2	Provides the support for OS PL/I PLICALLA and PLICALLB entry points. In link-edit steps, this library must precede SCEELKED if Enterprise PL/I applications use OS PL/I PLICALLA or PLICALLB as entry points.	X		
	SIBMMATH	Contains the stubs for old PL/I Version 2 Release 3 math library routines. In link-edit steps, this library must precede SCEELKED if PL/I for MVS and VM applications use OS PL/I PLICALLA or PLICALLB as entry points.	X		
	SIBMTASK	Provides the PL/I multitasking facility. In link-edit steps, this library must precede SCEELKED if PL/I multitasking facility is to be used.	X		

Chapter 3. Choosing your Language Environment runtime library access

Applications that require the runtime library provided by Language Environment can access the SCEERUN and SCEERUN2 data sets using:

- LNKLST
- STEPLIB

LNKLST

The Language Environment runtime libraries, SCEERUN, and SCEERUN2, can be placed in LNKLST. In addition, heavily-used modules can be placed in LPA. For more information see Appendix C, “Modules eligible for the link pack area,” on page 247.

STEPLIB

If the SCEERUN and SCEERUN2 data sets cannot be placed in LNKLST, you can STEPLIB the data sets for each application that requires them. One reason why the Language Environment runtime libraries are not to be placed in LNKLST might be that the pre-Language Environment runtime libraries (VS COBOL II, OS PL/I) are placed in LNKLST and your site has not completed the migration to Language Environment. See *z/OS Language Environment Runtime Application Migration Guide* for details.

Applications that currently STEPLIB to the SCEERUN data set to gain access to the runtime library provided by Language Environment, do not need to add the SCEERUN2 data set as part of their STEPLIB concatenation. In fact, since SCEERUN2 contains module names that do not intersect with any pre-Language Environment runtime library or any existing library, IBM recommends that SCEERUN2 be added to the LNKLST. This will not result in any adverse effects.

Chapter 4. Placing Language Environment modules in link pack and LIBPACK

Placing routines in the LPA/ELPA reduces the overall system storage requirement by making the routines shareable. Also, initialization/termination (init/term) time is reduced for each application, since load time decreases. For example, if Language Environment modules are not placed in LPA/ELPA, then under z/OS UNIX, every fork() call will require approximately 4 MB to be copied into the user address space.

The SCEERUN data set has many modules that are not reentrant, so you cannot place the entire data set in the Link Pack Area (LPALSTxx parmlib). There is a data set called SCEELPA that contains a subset of the SCEERUN modules: those that are reentrant, reside above the line, and are heavily used by z/OS itself. If you put the SCEERUN data set in the linklist (LNKLSTxx), you can place the SCEELPA data set in LPA list (LPALSTxx). Doing this will improve performance.

You cannot place the SCEERUN2 data set as part of a LPALSTxx because it is a PDSE. You must use the Dynamic LPA capability to move individual members of SCEERUN2 into the Link Pack Area.

You can also add additional modules to the LPA, using the Modify Link Pack Area (MLPA=) option at IPL. You can also use the Dynamic LPA capability (SET PROG=). Using the Dynamic LPA method avoids the performance degradation that occurs with the use of MLPA.

Choose which routines to put in the LPA/ELPA. See Appendix C, “Modules eligible for the link pack area,” on page 247 for a complete list of modules you may place in the LPA/ELPA.

Several members are installed in CEE.SCEESAMP for you to use as examples in creating your IEALPAnn or PROGxx member. Table 3 lists the members and their content.

Table 3. Language Environment sample IEALPAnn or PROGxx members in CEE.SCEESAMP

Member name	Description
CEEWLPA	All Language Environment base modules eligible for the LPA except callable service stubs. Uses Dynamic LPA.
EDCWLPA	All C/C++ component modules eligible for LPA from SCEERUN and SCEERUN2. Uses Dynamic LPA.
IGZWMLP4	All Language Environment COBOL component modules eligible for LPA.
IBMALLP2	All Language Environment PL/I component modules eligible for LPA
IBMPLPA1	MLPA macro for Enterprise PL/I
AFHWMLP2	All Language Environment Fortran modules eligible for LPA

If you want to load modules into the LPA, you do not need to place CEE.SCEERUN or CEE.SCEERUN2 in the LNKLSTmm member. However, if

CEE.SCEERUN or CEE.SCEERUN2 is not in the LNKLST nm member, you need to make modules that are not included in the link pack areas available to your application programs by copying the modules into a data set that can be either included in the LNKLST nm or used as a STEPLIB.

Using the entire CEE.SCEERUN or CEE.SCEERUN2 data set as a STEPLIB defeats the purpose of placing the modules in the LPA.

Shared storage considerations

- Modules you copy into another (non-LPA) data set are not automatically updated by SMP/E when you apply a service update. You must rerun your copy job after you apply service to Language Environment to make the updated modules available in the LNKLST nm data set or in the STEPLIB.
- Examine the lists carefully to make sure that you are installing the correct module for the national language support you have installed. Comments in CEEWLPA, EDCWLPA, and IBMALLP2 identify the Japanese modules. In IGZWMLP4, remove the module name IGZCMGEN if you do not want US English mixed-case to be in the LPA and add IGZCMGJA if Japanese is installed and you want it to be in the LPA.
- For more information about including modules in the LPA, see *z/OS MVS Initialization and Tuning Reference*.

Tailoring the Fortran LIBPACKs

The Fortran component of Language Environment is shipped with individual routines and with groupings of routines called LIBPACKs. A LIBPACK is a load module that contains individual library routines packaged together by the linkage editor into a single load module in order to reduce the time that would otherwise be needed to load the individual routines.

You might want to customize the Fortran LIBPACKs to:

- Shorten the load time for the Fortran LIBPACK by reducing its size
- Minimize the virtual storage required for an application by eliminating seldom-used routines from main storage
- Reduce the number of loads for application programs by adding frequently used routines to Fortran LIBPACKs
- Reduce the size of the contents of shared storage

Usage notes

The Fortran LIBPACKs are generally shared among several different applications and cannot be tuned for a specific application. Therefore, ideal Fortran LIBPACKs contain only library routines that are common to all application programs.

Choices to make now

You need to decide whether to modify the Fortran LIBPACKs. If you modify the Fortran LIBPACKs, you make a trade-off between use of storage and faster performance of application programs. See Table 4.

Table 4. Making the trade-off: Performance time versus storage use

Type of Fortran LIBPACK	Performance time	Storage use
Partially loaded	Slower because more routines are loaded individually	Less virtual and shared storage used

Table 4. Making the trade-off: Performance time versus storage use (continued)

Type of Fortran LIBPACK	Performance time	Storage use
Fully loaded	Faster because no routines loaded individually	More virtual and shared storage used

You can use the information in the following sections and the tables in “Language Environment Fortran component modules” on page 250 to decide which modules to include in your Fortran LIBPACKs.

Language Environment provides four Fortran LIBPACKs, which you can customize either during or following the installation of Language Environment.

- AFHPRNAG
- AFHPRNBG
- AFH5RENA
- AFH5RENB

After installation, each LIBPACK contains a default set of routines. You can remove many of the routines if their functions aren't used frequently at your site, or you can add others that you do use frequently.

Some examples

You can add or remove routines from the Fortran LIBPACKs to reflect the requirements of your location. For example, to include only the group of general routines that your location uses most often, eliminate unnecessary routines from the Fortran LIBPACK.

If you plan to put your Fortran LIBPACK into shared storage and your shared storage space is limited, consider reducing the size of your Fortran LIBPACKs. All modules eligible to be in the Fortran LIBPACKs are reentrant and are therefore eligible to be stored in the shared storage.

Listing the contents of Fortran LIBPACKs

Before tailoring your LIBPACKs, you might want to know their current structure, such as which MODs SMP/E expects to be combined into a particular load module, so that you can decide which ones to add or delete. Use SMP/E sample job AFHWLIST in the SCEESAMP data set to invoke the SMP/E LIST command to list the contents of your LIBPACKs.

Modifying the JCL for AFHWLIST

Perform the following steps to modify the JCL for AFHWLIST.

1. Change #GLOBALCSI to the data set name of your global CSI data set.

2. Change #TZONE to the name of your target zone.

3. Examine the LIBPACK names on the SMP/E LIST statement and remove the comments as appropriate.

When you are done, AFHWLIST should run with a condition code of 0.

Deleting routines from Fortran LIBPACKs

The sample jobs listed in Table 5 each contain SMP/E UCLIN and link-edit JCL that you can modify to delete routines to one of the Fortran LIBPACKs. The sample jobs are in target library CEE.SCEESAMP.

Table 5. SMP/E sample jobs for deleting routines from Fortran LIBPACKs

For applications link-edited with...	Use sample job...	To delete routines from LIBPACK...	Which is loaded ...
Language Environment	AFHWDERA	AFHPRNAG	Above 16 MB
Language Environment	AFHWDERB	AFHPRNBG	Below 16 MB
VS FORTRAN	AFHWDVRA	AFH5RENA	Above 16 MB
VS FORTRAN	AFHWDVRB	AFH5RENB	Below 16 MB

If the IBM-supplied LIBPACKs contain routines that your site does not use often, you can delete them using the following SMP/E sample jobs.

Steps for modifying the JCL to delete routines from a Fortran LIBPACK

Perform the following steps to modify the JCL to delete routines from a Fortran LIBPACK. These steps use the AFHWDERA, AFHWDERB, AFHWDVRA, and AFHWDVRB sample jobs.

1. Change #GLOBALCSI to the data set name of your global CSI data set.

2. Change #TZONE to the name of your target zone.

3. Modify the UCLIN step in the sample job to tell SMP/E to delete routines that you do not want to include in your tailored LIBPACK.
 - Remove the DELETE statement of any routine you want to include in your LIBPACK.
 - Remove the DELETE statement of any routine that is not currently in your LIBPACK.
 - If you run any of the sample jobs shown in Table 5 without modifying them, you receive a minimum LIBPACK without any optional modules.

4. The LINK-EDIT step performs the actual link edit of the tailored LIBPACK by replacing (deleting) the routines you have specified. The REPLACE statements you keep in the LINK-EDIT step must match the routines you specified in the UCLIN step.

When taking out the REPLACE records, ensure that all alias names (shown with indented REPLACE records) are removed too. For example, if you decide to remove AFHBCMVT, you need to remove AFHBCMVR as well.

5. Check the SYSLMOD DD statement to ensure that the data set name is correct.

When you are done, FHWDERA, AFHWDERB, AFHWDVRA, and AFHWDVRB should run with a condition code of 4. Unresolved external references for any optional modules not included in your LIBPACK are expected.

Adding routines to Fortran LIBPACKs

The sample jobs listed in Table 6 each contain SMP/E UCLIN and link-edit JCL that you can modify to add routines to one of the Fortran LIBPACKs. The sample jobs are in target library CEE.SCEESAMP.

Table 6. SMP/E sample jobs for adding routines to Fortran LIBPACKs

For applications link-edited with...	Use sample job...	To add routines to LIBPACK...	Which is loaded...
Language Environment	AFHWAERA	AFHPRNAG	Above 16 MB
Language Environment	AFHWAERB	AFHPRNBG	Below 16 MB
VS FORTRAN	AFHWAVRA	AFH5RENA	Above 16 MB
VS FORTRAN	AFHWAVRB	AFH5RENB	Below 16 MB

Note:

The jobs that add routines to the LIBPACKs add the versions of the routines that are in the target libraries.

If the IBM-supplied LIBPACKs exclude routines that your site uses often, you can add them using the SMP/E sample jobs that follow.

Steps for modifying the JCL for adding routines to a Fortran LIBPACK

Perform the following steps to modify the JCL for adding routines to a Fortran LIBPACK. These steps use the AFHWDERA, AFHWDERB, AFHWDVRA, and AFHWDVRB sample jobs.

1. Change #GLOBALCSI to the data set name of your global CSI data set.

2. Change #TZONE to the name of your target zone.

3. Modify the UCLIN step to tell SMP/E to add the routines you want to include in your tailored LIBPACK.
 - Remove the ADD statement for each routine you are not adding to your tailored LIBPACK.
 - If you run the sample jobs shown in Table 6 without modifying them, you receive a full LIBPACK, including all the required and optional LIBPACK modules.
 - If you attempt to add a routine that is already in the LIBPACK, you receive an SMP/E error message.

4. The LINK-EDIT step performs the actual link edit of the tailored LIBPACK by including the routines you specify.
The INCLUDE statements you keep in the LINK-EDIT step must match the routines you want to include in your tailored LIBPACK, regardless of whether you add the routine in the UCLIN step above or it is already in the LIBPACK.

5. Check the SYSLMOD DD statement to ensure that the data set name is correct.

When you are done, AFHWAERA, AFHWAERB, AFHWAVRA, and AFHWAVRB should run with a condition code of 0 if the LIBPACKs contain all of the optional modules. Otherwise, each of these jobs returns a condition code of 4; unresolved external references for any optional modules not included in the LIBPACKs are expected.

Where to place the tailored Fortran LIBPACKs

The sample jobs tailor the LIBPACKs and then use them to replace the LIBPACKs in the Language Environment target library SCEERUN. You could place them in another data set instead, provided that the LOADs issued during runtime can find them. The customized LIBPACKs must be found ahead of (in search-order sequence or in library concatenation), or instead of, those that were installed with the product. If you want to link edit a LIBPACK into an alternative library, modify and run only the LINK-EDIT step of the sample jobs.

Note: Because SMP/E is only aware of the load modules link-edited into the SCEERUN target library, SMP/E will not relink your LIBPACKs automatically when you apply service if you use an alternative library.

Part 2. Language Environment Customization: Runtime options, exits, and procedures

| This topic provides information about customizing Language Environment runtime
| options, using the CEEPRMxx parmlib member, customizing user exits,
| customizing cataloged procedures, using Language Environment under CICS,
| using Language Environment under IMS™, and customizing language-specific
| features.

Chapter 5. Customizing Language Environment runtime options

The default runtime option values IBM supplies with Language Environment may not suit the application programmers' needs at your site. A systems programmer can modify the IBM-supplied defaults on a system-level or region-level basis, which can save time by reducing the need to override the runtime option defaults as often. An application programmer can further refine these options based on individual program needs. When an application runs, runtime options are merged in a specific order of precedence to determine the actual values in effect. For more information about setting runtime options on an application-level, see *z/OS Language Environment Programming Guide* and *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

System-level defaults can be established through a member in the system parmlib called CEEPRMxx or with a SETCEE operator command. *Region-level* defaults can be established with a CEEROPT (AMODE 31) or CELQROPT (AMODE 64) load module created by invoking the CEEXOPT macro. For more information about the runtime options, default values, and syntax, see Chapter 6, "Language Environment runtime options," on page 45. You might not need to change most default values.

Table 7 summarizes the Language Environment runtime options, defaults, and recommended settings for applications running in CICS and non-CICS (for example, batch or IMS) environments. The recommended setting for some runtime options can vary, depending upon the language used to create the application or if multiple (Multi) languages are used in the environment. When a recommendation varies, the applicable settings for the languages are listed in the table. Also, the table identifies the runtime options that are not applicable (N/A) in either the CICS or non-CICS environment; Language Environment ignores these options if they are specified.

Table 7. Runtime options, defaults, and recommendations for Language Environment

Option	Non-CICS		CICS	
	Default	Recommended	Default	Recommended
ABPERC	NONE	NONE	N/A	N/A
ABTERMENC (see table note 1 on page 22)	ABEND	ABEND	ABEND	ABEND
AIXBLD	OFF	OFF	N/A	N/A
ALL31	ON	ON	ON	ON
ANYHEAP (see table notes 3 on page 22 and 11 on page 23)	16K,8K,ANY,FREE	16K,8K,ANY,FREE (C, COBOL, Multi, PL/I) 48K,8K,ANY,FREE (Fortran)	4K,4080,ANY,FREE	4K,4080,ANY,FREE
ARGPARSE (see table notes 4 on page 23 and 12 on page 23)	ARGPARSE	ARGPARSE	N/A	N/A
AUTOTASK	NOAUTOTASK	NOAUTOTASK	N/A	N/A
BELOWHEAP (see table note 11 on page 23)	8K,4K,FREE	8K,4K,FREE	4K,4080,FREE	4K,4080,FREE

Table 7. Runtime options, defaults, and recommendations for Language Environment (continued)

Option	Non-CICS		CICS	
	Default	Recommended	Default	Recommended
CBLOPTS (see table note 12 on page 23)	ON	ON	N/A	N/A
CBLPSHPOP	ON	N/A	ON	ON
CBLQDA	OFF	OFF	N/A	N/A
CEEDUMP	60,SYSOUT=*, FREE=END, SPIN=UNALLOC	60,SYSOUT=*, FREE=END, SPIN=UNALLOC	60,SYSOUT=*, FREE=END, SPIN=UNALLOC	60,SYSOUT=*, FREE=END, SPIN=UNALLOC
CHECK	ON	ON	ON	ON
COUNTRY (see table note 5 on page 23)	US	User-defined	US	User-defined
DEBUG	OFF	OFF	OFF	OFF
DEPTHCONDLMT	10	0	10	0
DYNDUMP	*USERID,NODYNAMIC, TDUMP	*USERID,NODYNAMIC, TDUMP	*USERID,NODYNAMIC, TDUMP	*USERID,NODYNAMIC, TDUMP
ENV (see table notes 4 on page 23 and 12 on page 23)	No default	User-defined	No default	User-defined
ENVAR	"	"	"	"
ERRCOUNT	0	0	0	0
ERRUNIT	6	6	N/A	N/A
EXECOPS (see table notes 4 on page 23 and 12 on page 23)	EXECOPS	EXECOPS	N/A	N/A
FILEHIST	ON	ON	N/A	N/A
FILETAG (see table note 12 on page 23)	NOAUTOCVT, NOAUTOTAG	NOAUTOCVT, NOAUTOTAG	N/A	N/A
FLOW (see table note 4 on page 23)	NOFLOW	FLOW	N/A	N/A
HEAP (see table notes 3 on page 22 and 11 on page 23)	32K,32K,ANY,KEEP, 8K,4K	32K,32K,ANY,KEEP, 8K,4K (C, COBOL, Multi, PL/I) 4K,4K,ANY,KEEP, 8K,4K (Fortran)	4K,4080,ANY,KEEP, 4K,4080	4K,4080,ANY,KEEP, 4K,4080
HEAP64	1M,1M,KEEP,32K,32K, KEEP,4K,4K,FREE	N/A	N/A	N/A
HEAPCHK (see table note 9 on page 23)	OFF,1,0,0,0,1024,0, 1024,0	OFF,1,0,0,0,1024,0, 1024,0	OFF,1,0,0,0,1024,0, 1024,0	OFF,1,0,0,0,1024,0, 1024,0
HEAPOOLS	OFF,8,10,32,10,128,10, 256,10,1024,10,2048,10	User-defined	OFF,8,10,32,10,128,10, 256,10,1024,10,2048,10	User-defined

Table 7. Runtime options, defaults, and recommendations for Language Environment (continued)

Option	Non-CICS		CICS	
	Default	Recommended	Default	Recommended
HEAPOOLS64	OFF,8,4000,32,2000,128, 700,256,350,1024,100, 2048,50,3072,50,4096, 50,8192,25,16384,10, 32768,5,65536,5	N/A	N/A	N/A
HEAPZONES (see table notes 4 on page 23, 9 on page 23 and 12 on page 23)	0,ABEND,0,ABEND	0,ABEND,0,ABEND	0,ABEND,0,ABEND	0,ABEND,0,ABEND
INFOMSGFILTER	OFF	OFF	OFF	OFF
INQPCOPN	ON	ON	N/A	N/A
INTERRUPT	OFF	OFF	N/A	N/A
IOHEAP64	1M,1M,FREE,12K,8K, FREE,4K,4K,FREE	N/A	N/A	N/A
LIBHEAP64	1M,1M,FREE,16K,8K, FREE,8K,4K,FREE	N/A	N/A	N/A
LIBSTACK	4K,4K,FREE	4K,4K,FREE	32,4080,FREE	32,4080,FREE
MSGFILE	SYSOUT,FBA,121,0, NOENQ	<i>ddname</i>	N/A	N/A
MSGQ	15	15	N/A	N/A
NATLANG	ENU	ENU	ENU	ENU
OCSTATUS	ON	ON	N/A	N/A
PAGEFRAMESIZE (see table notes 4 on page 23 and 12 on page 23)	4K,4K,4K	4K,4K,4K	N/A	N/A
PAGEFRAMESIZE64 (see table notes 4 on page 23 and 12 on page 23)	4K,4K,4K	4K,4K,4K	N/A	N/A
PC	OFF	OFF	N/A	N/A
PLIST (see table notes 4 on page 23 and 12 on page 23)	HOST	HOST	N/A	N/A
PLITASKCOUNT	20	20	N/A	N/A
POSIX	OFF	OFF	N/A	N/A
PROFILE	OFF,"	OFF,"	OFF,"	OFF,"
PRTUNIT	6	6	N/A	N/A
PUNUNIT	7	7	N/A	N/A
RDRUNIT	5	5	N/A	N/A
RECPAD	OFF	OFF	N/A	N/A
REDIR(see table notes 4 on page 23 and 12 on page 23)	REDIR	REDIR	N/A	N/A
RPTOPTS	OFF	OFF	OFF	OFF
RPTSTG	OFF	OFF	OFF	OFF
RTEREUS	OFF	OFF	N/A	N/A

Table 7. Runtime options, defaults, and recommendations for Language Environment (continued)

Option	Non-CICS		CICS	
	Default	Recommended	Default	Recommended
SIMVRD	OFF	OFF	N/A	N/A
STACK (see table note 11 on page 23)	128K,128K,ANY,KEEP,512K,128K	128K,128K,ANY,KEEP,512K,128K (C, Fortran, Multi, PL/I) 64K,64K,ANY,KEEP (COBOL)	4K,4080,ANY,KEEP,4K,4080	4K,4080,ANY,KEEP,4K,4080
STACK64	1M,1M,128M	N/A	N/A	N/A
STORAGE	NONE,NONE,NONE,0K	NONE,NONE,NONE,0K	NONE,NONE,NONE,0K	NONE,NONE,NONE,0K
TERMTHDACT	TRACE,,96	TRACE,,96 (C, Fortran, Multi, PL/I) UATRACE,,96 (COBOL)	TRACE,CESE,96	TRACE,CICSDDS,96 (C, Fortran, Multi, PL/I) UATRACE,CIDSDDS,96 (COBOL)
TEST	NOTEST(ALL,*,PROMPT,INSPPREF)	NOTEST(ALL,*,PROMPT,INSPPREF)	NOTEST(ALL,*,PROMPT,INSPPREF)	NOTEST(ALL,*,PROMPT,INSPPREF)
THREADHEAP	4K,4K,ANY,KEEP	4K,4K,ANY,KEEP	N/A	N/A
THREADSTACK	OFF,4K,4K,ANY,KEEP,128K,128K	OFF,4K,4K,ANY,KEEP,128K,128K	N/A	N/A
THREADSTACK64	OFF,1M,1M,128M	N/A	N/A	N/A
TRACE	OFF,4K,DUMP,LE=0	OFF,4K,DUMP,LE=0	OFF,4K,DUMP,LE=0	OFF,4K,DUMP,LE=0
TRAP	ON,SPIE	ON,SPIE	ON,SPIE	ON,SPIE
UPSI	00000000	00000000	00000000	00000000
USRHDLR (see table note 5 on page 23)	NOUSRHDLR	User-defined	NOUSRHDLR	User-defined
VCTRSAVE	OFF	OFF	N/A	N/A
XPLINK (see table notes 4 on page 23 and 12 on page 23)	OFF	OFF	N/A	N/A

Table notes:

1. When running with IMS, this setting ensures that IMS transactions are rolled back if errors occur in an application that is written in another Language Environment-enabled language; an abend causes IMS to roll back any database updates. When running a batch job, this setting ensures that a job step abends if errors occur in an application that is written in another language.
2. For PL/I, specify ALL31(OFF) for AMODE 24 programs. For COBOL, specify ALL31(OFF) if the applications contain one of the following:
 - A VS COBOL II NORES program (non-CICS program)
 - An OS/VS COBOL program (non-CICS program)
 - An AMODE 24 program

If you use ALL31(OFF), you must also specify STACK(,BELOW,); AMODE 24 programs usually require stack storage below the 16M line.
3. If your installation uses Fortran in a multi-language environment, use the recommended setting for Fortran.

4. You cannot specify this option as a system-level (CEEPRMxx parmlib member or SETCEE command) or region-level default. For more information about this option, see *z/OS Language Environment Programming Reference*.
5. There is no standard recommended value for this option; specify an appropriate value according to the needs of your installation.
6. If your installation uses COBOL in a multi-language environment, use the recommended option setting for COBOL.
7. Specify any name for the message output file. For Fortran applications, specify MSGFILE(FT06F001) to produce the same ddname as in VS Fortran.
8. For single-tasking PL/I applications, use the recommended Language Environment default. However, for multitasking PL/I applications, the following setting is recommended: THREADSTACK(4K,4K,BELOW,KEEP,,).
9. Specify this option only when developing and debugging applications.
10. To get behavior that is similar to the VS COBOL II runtime option WSCLEAR, use STORAGE(00,NONE,NONE,0K).
Do not use STORAGE(NONE,NONE,00,0K). Although it initializes variables for C and PL/I applications, serious performance degradation can occur. C and PL/I programs should be changed to initialize their own variables.
11. Acquiring a storage increment often involves a new storage obtain.
The increment size (4080 is recommended) should be 16 bytes less than the exact size of one or more pages to account for the 16-byte check zone that CICS applies to all storage obtain requests. This keeps Language Environment from obtaining an extra page of storage beyond the requested amount. This is important in CICS environments where storage below the line is especially constrained. The initial size in a CICS environment is part of a larger storage obtain that includes other storage required for Language Environment during initialization. Therefore, you can specify the initial size as exactly one or more pages, for example, 4K, 8K, and so on, without concern for acquiring an extra page.
12. You cannot specify this option with the CEEBXITA assembler user exit interface.

Creating system-level runtime option defaults with CEEPRMxx

Parmlib members are provided for specifying defaults for many system options. The CEEPRMxx parmlib member can be used for specifying system-level default runtime options that control various aspects of Language Environment. The CEEPRMxx parmlib member is identified during IPL by a CEE=xx statement, either in the IEASYSy parmlib member or in the IPL parameters. After IPL, the operator can do the following tasks:

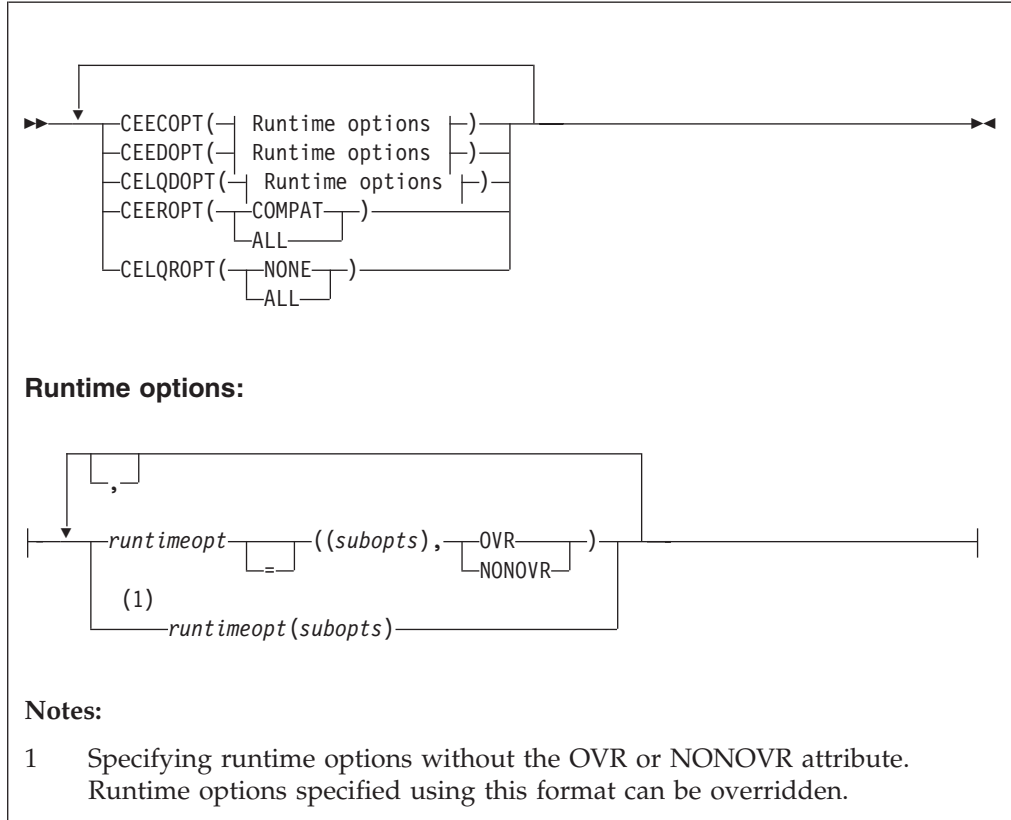
- Change the active CEEPRMxx parmlib member with the SET CEE=xx command.
- Change individual runtime options using the SETCEE command.
- Display current runtime option settings with the D CEE command.
- Clear all the system-level default runtime options and keywords using the SETCEE CLEAR command.
- Check existing CEEPRMxx parmlib members for valid syntax.

Using this support is not required, so the default IEASYS00 parmlib member does not specify a CEEPRMxx parmlib member. If you want to use this support, a sample CEEPRM00 member is included in CEE.SCEESAMP.

CEEPRMxx parmlib member

Use the CEEPRMxx parmlib member to set system-level default runtime options.

Syntax



CEECOPT

The options group used to specify runtime options for CICS environments.

CEEDOPT

The options group used to specify runtime options for non-CICS environments excluding AMODE 64 environments.

CELQDOPT

The options group used to specify runtime options for AMODE 64 environments.

CEEROPT

Indicates whether region-level runtime options should be used in a non-CICS or non-LRR environment.

COMPAT

Attempt a load and use of **CEEROPT** only in CICS or LRR environments. This is the default behavior.

ALL

Attempt a load and use of **CEEROPT** in all AMODE 31 and AMODE 24 environments.

CELQROPT

Indicates whether region-level runtime options should be used in an AMODE 64 environment.

NONE

Do not attempt a load or use of **CELQROPT** in AMODE 64 environments. This is the default behavior.

ALL

Attempt a load and use of **CELQROPT** in all AMODE 64 environments.

runtimeopt

The name of the runtime option to change.

subopts

The suboption values for the specified runtime option to change.

OVR

Specifies that the option can be overridden. This is the default setting.

NONOVR

Specifies that the option cannot be overridden, which can be used to enforce runtime options critical to the Language Environment operating environment.

Once a runtime option is specified as nonoverrideable with the **NONOVR** attribute, it cannot later be overridden. This includes later specification in the same parmlib member or a **SETCEE** command. To remove the nonoverrideable setting, use the **SETCEE CLEAR** operator command, or the **SET CEE** command with a parmlib member that does not mark the runtime option as nonoverrideable.

Note: The runtime options **AIXBLD**, **DEBUG**, **FILEHIST**, **INQPCOPN**, **OCSTATUS**, **PC**, **RTEREUS** and **SIMVRD** require an **ON** or **OFF** suboption to be specified when using the **OVR** or **NONOVR** attribute. For example, **AIXBLD=((),OVR)** results in an error and the option is ignored.

Usage notes

- The options for each group are saved independently to allow **CEEDOPT** to specify **NATLANG(ENU)** while **CEEEOPT** can specify **NATLANG(JPN)**.
- Enter values in uppercase, lowercase, or mixed case. The system converts the input to uppercase, except for values enclosed in single quotation marks, which are processed without changing the case.
- Commas are required between suboptions and before an **OVR** or **NONOVR** attribute.
- Commas are allowed between options.
- Use blanks or commas as delimiters. Multiple blanks are interpreted as a single blank. Blanks are allowed between parameters and values.
Restriction: Blanks are not allowed within the required **=((** or **((** delimiter for runtime options that specify an **OVR** or **NONOVR** attribute.
- More than one option can be specified on a line. An option can be continued on multiple lines.
- Comments can appear in columns 1-71 and must begin with **"/*** and end with ***/**.
- Nested comments are not supported.
- When **CEEROPT(ALL)** is in effect, an attempt is made to load a **CEEROPT** module during Language Environment initialization. When no **CEEROPT** can be

found, there is potential performance overhead, especially for applications or transaction servers that repeatedly initialize Language Environment.

- You can specify a group or keyword more than once in a single member.
- A runtime option can be repeated within a parmlib member, group, or can appear in multiple members when more than one member is specified. Runtime options are processed in the order in which they appear. The suboptions are saved for each occurrence. When suboptions are repeated, the last occurrence is used. The last occurrence of the runtime option is identified as the runtime options source in the runtime options report.

Restriction: Only the last occurrence of the ENVAR runtime option is saved. If multiple members are used, and ENVAR appears in the same runtime options group in more than one member, the last one found is saved.

- When the runtime options are merged during the initialization of a Language Environment application, errors might be reported if any system-level or region-level defaults were marked nonoverridable (NONOVR). These messages are displayed for every application. Under CICS, the messages are displayed for the first transaction only.
- If RPTOPTS(ON) is in effect at runtime, the Language Environment runtime options report is displayed, and the Last Where Set column will identify any options that were set by a CEEPRMxx parmlib member. For a sample of the runtime options report, see *z/OS Language Environment Debugging Guide*.

Note: CICS TS 3.1 and higher supports XPLINK programs in a CICS environment. The options specified in the CEEDOPT group are used for these programs.

Examples

The following example shows the IBM-supplied sample of the CEEPRM00 parmlib member provided in the CEE.SCEESAMP data set. All valid groups, options, suboptions with their default values, and keywords are coded in the sample within comment characters. You must remove the comment characters from the groups, options, or keywords that you want to use.

```

/*****/
/* CEEPRM00 - Sample Language Environment parmlib member for */
/*      runtime options.                                     */
/*                                                         */
/* LICENSED MATERIALS - PROPERTY OF IBM                    */
/*                                                         */
/* 5650-ZOS                                                 */
/*                                                         */
/* COPYRIGHT IBM CORP. 2005, 2012                          */
/*                                                         */
/* ALL RIGHTS RESERVED                                     */
/*                                                         */
/* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,           */
/* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP        */
/* SCHEDULE CONTRACT WITH IBM CORP.                        */
/*                                                         */
/* STATUS = HLE7790                                         */
/*                                                         */
/*                                                         */
/* This sample parmlib member contains the IBM-supplied default */
/* runtime options that are valid at the system level. The defaults */
/* can be overridden using the options groups CEEDOPT, CEEOPT, and */
/* CELQDOPT.                                               */
/*                                                         */
/* This sample also contains the default values for the CEEOPT and */
/* CELQROPT keywords.                                     */
/*                                                         */
/*****/

```

```

/*                                                                    */
/* Syntax for options:                                                */
/*                                                                    */
/*   group_name( option_1, option_2,                                  */
/*               option_3, option_4 )                                */
/*                                                                    */
/* Where:                                                              */
/*   group_name is CEEDOPT, CEECOPT or CELQDOPT.                    */
/*   option_x is any option valid at the system level.              */
/*                                                                    */
/*                                                                    */
/* Syntax for keywords:                                              */
/*                                                                    */
/*   CEEROPT( value ) - Where value is ALL or COMPAT                */
/*   CELQROPT( value ) - Where value is ALL or NONE                  */
/*                                                                    */
/*                                                                    */
/* All valid options and keywords are listed but commented out.     */
/* To include an option you must edit this file (or a copy) and     */
/* remove the comment delimiters around the options to be used.     */
/* It is not necessary to uncomment all options.                    */
/*                                                                    */
/*                                                                    */
/* Notes:                                                             */
/* * Comments and blank lines are allowed for readability.         */
/* * Individual options must be separated by a comma or a blank     */
/* * There can be more than one option on a line.                   */
/* * Mixed case is allowed.                                         */
/* * Individual options can be specified as overrideable (OVR) or  */
/*   nonoverrideable (NONOVR).                                       */
/*                                                                    */
/*                                                                    */
/*****                                                                    */
/*****                                                                    */
/* 31 bit non-CICS option group                                       */
/*****                                                                    */
/*****                                                                    */
/*CEEDOPT(                                                            */
/*   ABPERC=((NONE),OVR),                                            */
/*   ABTERMENC=((ABEND),OVR),                                        */
/*   AIXBLD=((OFF),OVR),                                           */
/*   ALL31=((ON),OVR),                                             */
/*   ANYHEAP=((16K,8K,ANYWHERE,FREE),OVR),                         */
/*   BELOWHEAP=((8K,4K,FREE),OVR),                                  */
/*   CBLOPTS=((ON),OVR),                                           */
/*   CBLPSHPOP=((ON),OVR),                                         */
/*   CBLQDA=((OFF),OVR),                                           */
/*   CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR),           */
/*   CHECK=((ON),OVR),                                              */
/*   COUNTRY=((US),OVR),                                           */
/*   DEBUG=((OFF),OVR),                                            */
/*   DEPTHCONDLMT=((10),OVR),                                       */
/*   DYNDUMP=((*USERID,NODYNAMIC,TDUMP),OVR),                      */
/*   ENVAR=('',OVR),                                               */
/*   ERRCOUNT=((0),OVR),                                           */
/*   ERRUNIT=((6),OVR),                                            */
/*   FILEHIST=((ON),OVR),                                          */
/*   FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR),                          */
/*   HEAP=((32K,32K,ANYWHERE,KEEP,8K,4K),OVR),                   */
/*   HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR),                   */
/*   HEAPPOOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048,      */
/*             10,0,10,0,10,0,10,0,10,0,10,0,10,0,10,0),OVR),   */
/*   INFMSGFILTER=((OFF,,,),OVR)                                    */

```

```

/*      INQPCOPN=((ON),OVR),          */
/*      INTERRUPT=((OFF),OVR),        */
/*      LIBSTACK=((4K,4K,FREE),OVR),  */
/*      MSGFILE=((SYSOUT,FBA,121,0,NOENQ),OVR), */
/*      MSGQ=((15),OVR),              */
/*      NATLANG=((ENU),OVR),          */
/*      NOAUTOTASK=(OVR),            */
/*      NOTEST=((ALL,*,PROMPT,INSPREF),OVR), */
/*      NOUSRHDLR=((),OVR),          */
/*      OCSTATUS=((ON),OVR),          */
/*      PC=((OFF),OVR),               */
/*      PLITASKCOUNT=((20),OVR),     */
/*      POSIX=((OFF),OVR),            */
/*      PROFILE=((OFF,''),OVR),       */
/*      PRTUNIT=((6),OVR),            */
/*      PUNUNIT=((7),OVR),           */
/*      RDRUNIT=((5),OVR),            */
/*      RECPAD=((OFF),OVR),           */
/*      RPTOPTS=((OFF),OVR),          */
/*      RPTSTG=((OFF),OVR),           */
/*      RTEREUS=((OFF),OVR),          */
/*      SIMVRD=((OFF),OVR),           */
/*      STACK=((128K,128K,ANYWHERE,KEEP,512K,128K),OVR), */
/*      STORAGE=((NONE,NONE,NONE,0K),OVR), */
/*      TERMTHDACT=((TRACE,,96),OVR), */
/*      THREADHEAP=((4K,4K,ANYWHERE,KEEP),OVR), */
/*      THREADSTACK=((OFF,4K,4K,ANYWHERE,KEEP,128K,128K),OVR), */
/*      TRACE=((OFF,4K,DUMP,LE=0),OVR), */
/*      TRAP=((ON,SPIE),OVR),         */
/*      UPSI=((00000000),OVR),        */
/*      VCTRSAVE=((OFF),OVR),         */
/*      XUFLOW=((AUTO),OVR)           */
/*      )                             */
/*****/
/* 31 bit CICS option group          */
/* The following options are ignored in CICS: */
/* - ABPERC                          - OCSTATUS */
/* - AIXBLD                           - PC */
/* - AUTOTASK                          - PLITASKCOUNT */
/* - CBLOPTS                           - POSIX */
/* - CBLQDA                            - PRTUNIT */
/* - DYNDUMP                           - PUNUNIT */
/* - ERRUNIT                           - RDRUNIT */
/* - FILEHIST                          - RECPAD */
/* - FILETAG                           - RTEREUS */
/* - INQPCOPN                          - SIMVRD */
/* - INTERRUPT                         - THREADHEAP */
/* - MSGFILE                           - THREADSTACK */
/* - MSGQ                              - VCTRSAVE */
/*                                     */
/*****/
/*CEECOPT(                            */
/*      ABTERMENC=((ABEND),OVR),       */
/*      ALL31=((ON),OVR),              */
/*      ANYHEAP=((4K,4080,ANYWHERE,FREE),OVR), */
/*      BELOWHEAP=((4K,4080,FREE),OVR), */
/*      CBLPSHPOP=((ON),OVR),         */
/*      CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR), */
/*      CHECK=((ON),OVR),              */
/*      COUNTRY=((US),OVR),           */
/*      DEBUG=((OFF),OVR),            */
/*      DEPTHCONDLMT=((10),OVR),      */
/*      ENVAR=(('),OVR),              */
/*      ERRCOUNT=((0),OVR),          */
/*      HEAP=((4K,4080,ANYWHERE,KEEP,4K,4080),OVR), */
/*      HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR), */
/*      HEAPPOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048),OVR)

```

```

/*      10,0,10,0,10,0,10,0,10,0,10,0,10),OVR), */
/*      INFMSGFILTER=((OFF,,,),OVR), */
/*      LIBSTACK=((32,4080,FREE),OVR), */
/*      NATLANG=((ENU),OVR), */
/*      NOTEST=((ALL,*,PROMPT,INSPREF),OVR), */
/*      NOUSRHLDR=((),OVR), */
/*      PROFILE=((OFF,''),OVR), */
/*      RPTOPTS=((OFF),OVR), */
/*      RPTSTG=((OFF),OVR), */
/*      STACK=((4K,4080,ANYWHERE,KEEP,4K,4080),OVR), */
/*      STORAGE=((NONE,NONE,NONE,0K),OVR), */
/*      TERMTHDACT=((TRACE,CESE,96),OVR), */
/*      TRACE=((OFF,4K,DUMP,LE=0),OVR), */
/*      TRAP=((ON,SPIE),OVR), */
/*      UPSI=((00000000),OVR), */
/*      XUFLOW=((AUTO),OVR) */
/*      ) */
/*****
/* 64 bit options group */
/*****
/*CELQDOPT(
/*      CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR), */
/*      DYNDUMP=((*USERID,NODYNAMIC,TDUMP),OVR), */
/*      ENVAR=(('),OVR), */
/*      FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR), */
/*      HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR), */
/*      HEAPOOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048, */
/*      10,0,10,0,10,0,10,0,10,0,10,0,10),OVR), */
/*      HEAPOOLS64=((OFF,8,4000,32,2000,128,700,256,350, */
/*      1024,100,2048,50,3072,50,4096,50,8192,25,16384,10, */
/*      32768,5,65536,5),OVR) */
/*      HEAP64=((1M,1M,KEEP,32K,32K,KEEP,4K,4K,FREE), */
/*      INFMSGFILTER=((OFF,,,),OVR), */
/*      IOHEAP64=((1M,1M,FREE,12K,8K,FREE,4K,4K,FREE),OVR), */
/*      LIBHEAP64=((1M,1M,FREE,16K,8K,FREE,8K,4K,FREE),OVR), */
/*      NATLANG=((ENU),OVR), */
/*      NOTEST=((ALL,*,PROMPT,INSPREF),OVR), */
/*      POSIX=((OFF),OVR), */
/*      PROFILE=((OFF,''),OVR), */
/*      RPTOPTS=((OFF),OVR), */
/*      RPTSTG=((OFF),OVR), */
/*      STACK64=((1M,1M,128M),OVR), */
/*      STORAGE=((NONE,NONE,NONE),OVR), */
/*      THREADSTACK64=((OFF,1M,1M,128M),OVR), */
/*      TERMTHDACT=((TRACE,,96),OVR), */
/*      TRACE=((OFF,,DUMP,LE=0),OVR), */
/*      TRAP=((ON,SPIE),OVR) */
/*      ) */

/*****
/* Keywords */
/*****
/*CEEROPT( COMPAT )
/*CELQROPT( NONE )

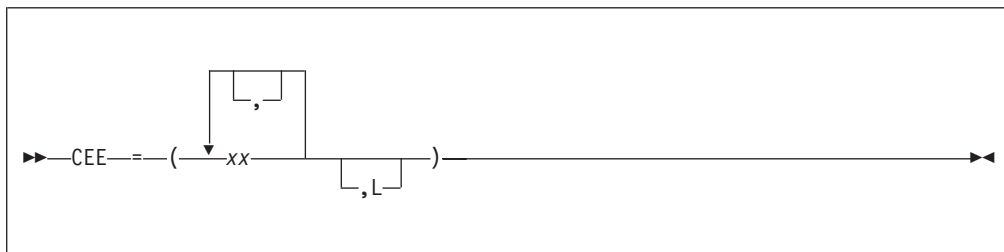
```

CEE= statement at IPL

Use the CEE=xx statement to specify CEEPRMxx parmlib members during system IPL. The CEE=xx statement can be specified in the IEASYSsy parmlib member or in the IPL parameters.

CEE= statement

Syntax



xx Two alphanumeric characters that specify a CEEPRM*xx* parmlib member.

L Indicates that the system will write the statements from the associated parmlib members to the operator's console during system initialization.

Usage notes

- When multiple members are specified, they are processed in the order specified.
- When the runtime options are merged during the initialization of a Language Environment application, errors might be reported if any system-level or region-level defaults were marked nonoverridable (NONOVR). These messages are displayed for every application. Under CICS, the messages are displayed for the first transaction only.

Examples

During IPL, you can specify CEEPRM*xx* parmlib members by using one of the methods shown in these examples:

- To specify a single parmlib member on the system parameters entered at IPL:

```
R 0,SYSP=yy,CEE=xx
```
- To specify a single parmlib member in the IEASYSy parmlib member:

```
ALLOC=01  
APF=PX,  
CEE=xx,  
CLOCK=00,  
CLPA, CMD=PX,
```
- To specify multiple parmlib members in the IEASYSy parmlib member:

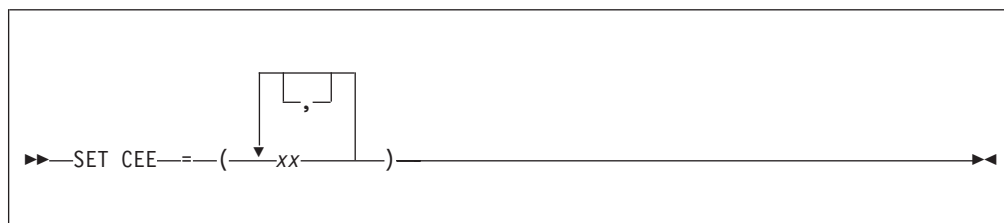
```
ALLOC=01  
APF=PX,  
CEE=(xx,zz),  
CLOCK=00,  
CLPA, CMD=PX,
```
- To specify multiple parmlib members with the L option on the system parameters entered at IPL:

```
R 0,SYSP=yy,CEE=(xx,zz,L)
```

SET CEE command

Use the SET CEE command to change the active parmlib member after IPL. The SET CEE command parses the CEEPRM*xx* parmlib member and replaces the runtime options and keywords with the contents of the new member.

Syntax



`xx` Two alphanumeric characters that specify a CEEPRMxx parmlib member.

Usage notes

- If you specify only one member, parentheses are optional.
- If you specify two or more members, parentheses are required.
- Changing the system-level default runtime options with the SET CEE command does not affect any currently initialized environments on the system. When applications go through Language Environment initialization, the new runtime option values are used for that application.
- When the runtime options are merged during the initialization of a Language Environment application, errors might be reported if any system-level or region-level defaults were marked nonoverridable (NONOVR). These messages are displayed for every application. Under CICS, the messages are displayed for the first transaction only.

Examples

The following example shows how to change the active parmlib member to CEEPRMJ1:

```
SET CEE=J1
```

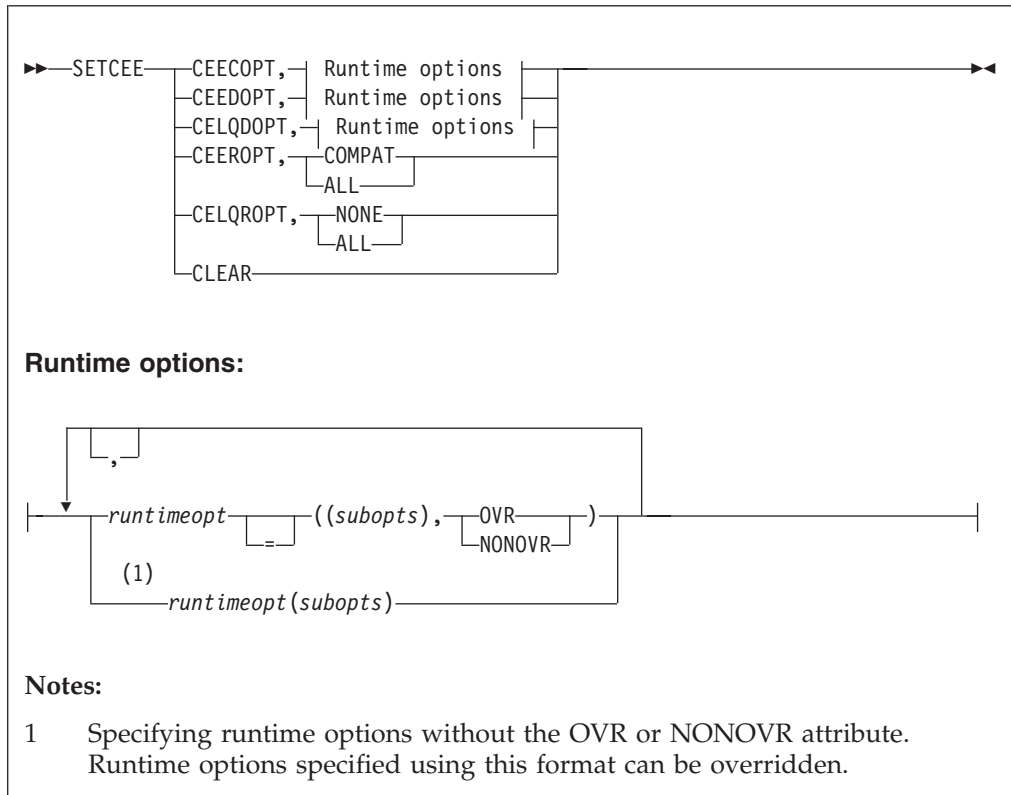
The following example shows how to change the active parmlib members to CEEPRMJC and CEEPRMJM:

```
SET CEE=(JC,JM)
```

SETCEE command

Use the SETCEE command to change individual runtime options or keywords.

Syntax



CEECOPT

The options group used to specify runtime options for CICS environments.

CEEDOPT

The options group used to specify runtime options for non-CICS environments excluding AMODE 64 environments.

CELQDOPT

The options group used to specify runtime options for AMODE 64 environments.

CEEROPT

Indicates whether region-level runtime options should be used in a non-CICS or non-LRR environment.

COMPAT

Attempt a load and use of **CEEROPT** only in CICS or LRR environments. This is the default behavior.

ALL

Attempt a load and use of **CEEROPT** in all AMODE 31 and AMODE 24 environments.

CELQROPT

Indicates whether region-level runtime options should be used in an AMODE 64 environment.

NONE

Do not attempt a load or use of **CELQROPT** in AMODE 64 environments. This is the default behavior.

ALL

Attempt a load and use of **CELQROPT** in all AMODE 64 environments.

CLEAR

Clear all the system-level default runtime options and keywords that were set using the SETCEE or SET CEE commands.

runtimeopt

The name of the runtime option to change.

subopts

The suboption values for the specified runtime option to change.

OVR

Specifies that the option can be overridden. This is the default setting.

NONOVR

Specifies that the option cannot be overridden, which can be used to enforce runtime options critical to the Language Environment operating environment.

Once a runtime option is specified as nonoverrideable with the NONOVR attribute, it cannot later be overridden. This includes later specification in the same parmlib member or a SETCEE command. To remove the nonoverrideable setting, use the SETCEE CLEAR operator command, or the SET CEE command with a parmlib member that does not mark the runtime option as nonoverrideable.

Note: The runtime options AIXBLD, DEBUG, FILEHIST, INQPCOPN, OCSTATUS, PC, RTEREUS and SIMVRD require an ON or OFF suboption to be specified when using the OVR or NONOVR attribute. For example, AIXBLD=((),OVR) results in an error and the option is ignored.

Usage notes

- Any group, keyword, or runtime option that can be specified in a CEEPRMxx parmlib member is valid.
- A maximum of 126 characters is allowed for each command. There is no continuation.
- You can use one SETCEE command to modify multiple runtime options within a single group.
- Enter values in uppercase, lowercase, or mixed case. The system converts the input to uppercase, except for values enclosed in single quotation marks, which are processed without changing the case.
- Commas are required between options, suboptions and before an OVR or NONOVR attribute.
- Blanks are not allowed within the required =((or ((delimiter for runtime options that specify an OVR or NONOVR attribute.
- To synchronize the setting of multiple runtime options or keywords, use the SET CEE command to process additional parmlib members. For more information, see “SET CEE command” on page 30.
- When CEEROPT(ALL) is in effect, an attempt is made to load a CEEROPT module during Language Environment initialization. When no CEEROPT can be found, there is potential performance overhead, especially for applications or transaction servers that repeatedly initialize Language Environment.
- Changing the system-level default runtime options with the SETCEE command does not affect any currently initialized environments on the system. When applications go through Language Environment initialization, the new runtime option values are used for that application.

SETCEE

- When the runtime options are merged during the initialization of a Language Environment application, errors might be reported if any system-level or region-level defaults were marked nonoverridable (NONOVR). These messages are displayed for every application. Under CICS, the messages are displayed for the first transaction only.
- If RPTOPTS(ON) is in effect at runtime, the Language Environment runtime options report is displayed, and the Last Where Set column will identify any options that were set by a CEEPRMxx parmlib member. For a sample of the runtime options report, see *z/OS Language Environment Debugging Guide*.
- When the SETCEE command completes, Language Environment produces message CEE3743I stating that the SETCEE command completed successfully.

Examples

The following example shows variations of using the SETCEE command to set individual runtime options:

```
SETCEE CEEDOPT,POSIX=((ON),OVR)
SETCEE CEECOPT,DEBUG=((OFF),NONOVR)
SETCEE CELQDOPT,HEAP64(1M),IOHEAP64(1M,1M)
```

The following example shows how to attempt a load and use CEEROPT in all AMODE 31 and AMODE 24 environments:

```
SETCEE CEEROPT,ALL
```

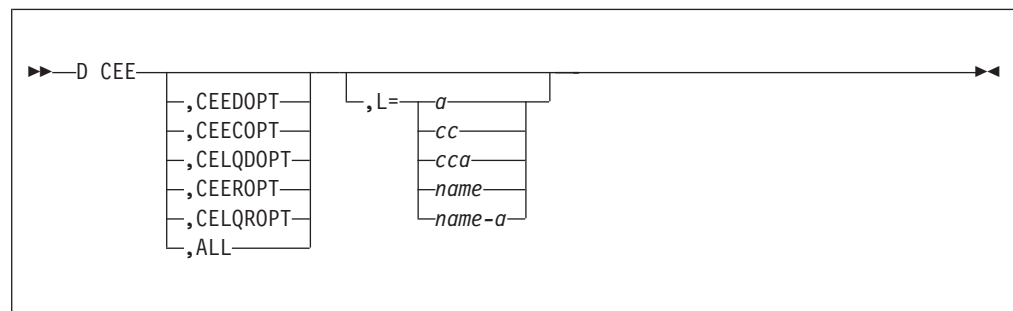
The following example shows how to clear all system-level default runtime options and keywords:

```
SETCEE CLEAR
```

D CEE command

Use the D CEE command to display the values that were set in the current CEEPRMxx parmlib members and by the SETCEE command.

Syntax



CEEDOPT

Displays the current list of system-level default runtime options for z/OS batch.

CEECOPT

Displays the current list of system-level default runtime options for CICS environments.

CELQDOPT

Displays the current list of system-level default runtime options for AMODE 64 environments.

CEEROPT

Displays the current value for the CEEROPT keyword.

CELQROPT

Displays the current value for the CELQROPT keyword.

ALL

Specifies that all keywords and groups with their respective option values should be displayed.

L Specifies where the display is presented.

a Specifies the display area.

cc Specifies the console. You must specify a decimal number from 1 to 99.

cca

Specifies the console and display area.

name

Specifies the console name.

name-a

Specifies the console name and display area.

Usage notes

None.

Examples

- If the SET CEE command is: set cee=(01,pv)

```
d cee
CEE3744I 10.55.33 DISPLAY
CEE=(01,PV)

d cee,ceedopt
CEE3745I 10.55.59 DISPLAY CEEDOPT
CEE=(01,PV)
LAST WHERE SET                OPTION
-----
PARMLIB(CEEPRM01)              ENVAR("verify=1 2 3")
PARMLIB(CEEPRMPV)              HEAP(4194304,5242880,ANYWHERE,KEEP,
                                16384,16384)
PARMLIB(CEEPRMPV)              POSIX(ON)
PARMLIB(CEEPRM01)              PROFILE(OFF,"XXX")
PARMLIB(CEEPRM01)              RPTOPTS(ON)
PARMLIB(CEEPRMPV)              STORAGE(AA,BB,NONE,0)
PARMLIB(CEEPRMPV)              THREADHEAP(8192,10240,ANYWHERE,KEEP)
```

- If the SETCEE command is: setcee ceedopt,rptopts(off)

```
d cee,ceedopt
CEE3745I 10.59.33 DISPLAY CEEDOPT
CEE=(01,PV)
LAST WHERE SET                OPTION
-----
PARMLIB(CEEPRM01)              ENVAR("verify=1 2 3")
PARMLIB(CEEPRMPV)              HEAP(4194304,5242880,ANYWHERE,KEEP,
                                16384,16384)
PARMLIB(CEEPRMPV)              POSIX(ON)
PARMLIB(CEEPRM01)              PROFILE(OFF,"XXX")
```

D CEE

```
SETCEE command          RPTOPTS(OFF)
PARMLIB(CEEPRMPV)      STORAGE(AA,BB,NONE,0)
PARMLIB(CEEPRMPV)      THREADHEAP(8192,10240,ANYWHERE,KEEP)
```

- If the SETCEE command is: setcee clear

```
d cee,all
CEE3745I 12.50.54 DISPLAY CEEDOPT
NO MEMBERS SPECIFIED
LAST WHERE SET          OPTION
-----
```

```
CEE3745I 12.50.54 DISPLAY CEECOPT
NO MEMBERS SPECIFIED
LAST WHERE SET          OPTION
-----
```

```
CEE3745I 12.50.54 DISPLAY CELQDOPT
NO MEMBERS SPECIFIED
LAST WHERE SET          OPTION
-----
```

```
CEE3745I 13.40.06 DISPLAY CEEROPT
NO MEMBERS SPECIFIED
NO CEEROPT KEYWORD SPECIFIED
CEE3745I 13.40.06 DISPLAY CELQROPT
NO MEMBERS SPECIFIED
NO CELQROPT KEYWORD SPECIFIED
```

- If the SET CEE command is: set cee=(mc)

```
d cee,ceeropt
CEE3745I 16.17.23 DISPLAY CEEROPT
CEE=(MC)
PARMLIB(CEEPRMMC) CEEROPT (COMPAT)
```

- If the SETCEE command is: setcee celqropt,all

```
d cee,celqropt
CEE3745I 16.14.52 DISPLAY CELQROPT
CEE=(MC)
SETCEE COMMAND    CELQROPT(ALL)
```

CEEPRMCC - syntax checking under z/OS batch

CEEPRMxx parmlib members can be syntactically checked for errors under z/OS batch. Before calling the syntax checker, a CEEPRMxx parmlib member must be created and placed in a PDS or PDSE. You can find the format and requirements for creating the parmlib member in “CEEPRMxx parmlib member” on page 24.

The CEEPRMCC program reads and parses a CEEPRMxx parmlib member for syntax errors, and displays the runtime options report if no errors are found. The runtime options report only displays options that are specified inside the CEEPRMxx parmlib members.

Syntax

The CEEPRMCC program expects the following inputs:

- The PARM parameter of the EXEC job control statement to select one or more CEEPRMxx parmlib members:

```
// PARM='CEE=(xx,yy,...,nn)'
```

The two alphanumeric characters, xx,yy,...,nn, are the suffix of the CEEPRMxx parmlib members to be checked. Embedded blanks are not allowed within the PARM.

- An optional CEEPRMCK DD statement to specify the data set where CEEPRMxx parmlib members are located:

```
//CEEPRMCK DD DSN=MEENAK.SYSTEM.PARMLIB,DISP=SHR
```

If no DD is specified, the CEEPRMCC program uses the default data set SYS1.PARMLIB.

Usage notes

An input data set must be a fixed record format and a record length of 80.

Return codes

The possible return codes are as follows:

- | | |
|------|--|
| 0 | Successful completion. |
| 4 | No members were specified. |
| 8 | Input members were not specified or not valid. |
| 12 | A closing parenthesis was missing when specifying input members. |
| 16 | One or more chars were found after the closing parenthesis of the input members. |
| 20 | One or more embedded blanks were found. |
| 24 | SYS1.PARMLIB allocation to the CEEPRMCK DD failed. |
| 28 | The specified data set had one or more incorrect attributes. |
| 32 | The specified data set did not exist. |
| 1004 | The input members string contained a single char suffix. |
| 1008 | The specified members could not be read. |
| 1012 | The specified members had one or more syntax errors. |
| 1016 | The specified member has three or more char suffix. |

Examples

- The following example shows how to check the CEEPRMJM parmlib member, which resides in the MEENAK.SYSTEM.PARMLIB data set:

```
//CEEPRMCJ EXEC PGM=CEEPRMCC,  
// PARM='CEE=(JM)'  
//CEEPRMCK DD DSN=MEENAK.SYSTEM.PARMLIB,DISP=SHR
```

- If there are syntax errors, no runtime options report will be displayed. Error messages will be written to the Language Environment message file. CEE3761I will be followed by other existing error messages related to syntax errors in CEEPRMxx parmlib members and end with CEE3762I, for example:

```
CEE3761I The following messages pertain to the call to the Language  
Environment Parmlib checker.  
CEE3731I The following messages pertain to the system default  
runtime options in the CEEDOPT in CEEPRMME.  
CEE3616I The string 'NNE' was not a valid or supported suboption  
of the runtime option STORAGE in this release.  
CEE3762I The Language Environment Parmlib checker has completed.
```

- The following example shows sample output when no errors are found:

```
CEE3762I The Language Environment Parmlib checker has completed.  
  
CEE3745I 11.14.01 Display CEEDOPT  
CEE=(ME)
```

CEEPRMCC - z/OS batch

```

LAST WHERE SET          OPTION
-----
PARMLIB(CEEPRMME)      POSIX(OFF)
PARMLIB(CEEPRMME)      STORAGE(NONE,NONE,NONE,0)

CEE3745I 11.14.01 Display CEECOPT
CEE=(ME)

LAST WHERE SET          OPTION
-----
PARMLIB(CEEPRMME)      STORAGE(NONE,NONE,20,2048)

CEE3745I 11.14.01 Display CELQDOPT
CEE=(ME)

LAST WHERE SET          OPTION
-----
PARMLIB(CEEPRMME)      POSIX(OFF)
PARMLIB(CEEPRMME)      STORAGE(NONE,NONE,30,3072)

CEE3745I 11.14.01 Display CEEROPT
CEE=(MS)

PARMLIB(CEEPRMMS) CEEROPT(ALL)

CEE3745I 11.14.01 Display CELQROPT
CEE=(MS)

PARMLIB(CEEPRMMS) CELQROPT(NONE)

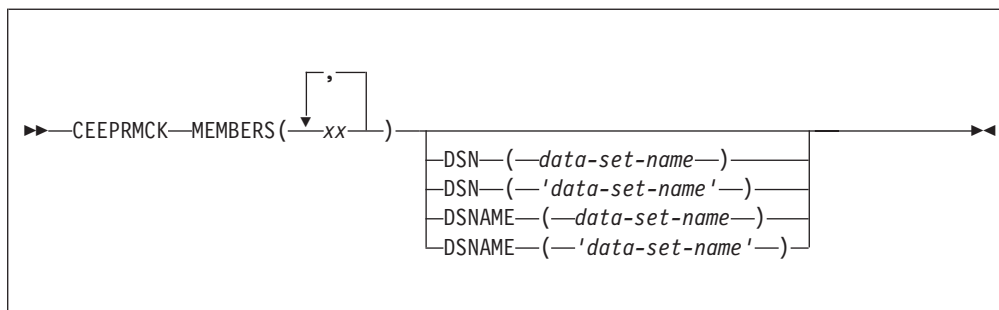
```

CEEPRMCK - syntax checking under TSO/E

CEEPRMxx parmlib members can be syntactically checked for errors under TSO/E. Before calling the syntax checker, a CEEPRMxx parmlib member must be created and placed in a PDS or PDSE. You can find the format and requirements for creating the parmlib member in “CEEPRMxx parmlib member” on page 24.

The CEEPRMCK program reads and parses the CEEPRMxx parmlib member for syntax errors, and displays the runtime options report if no errors are found. The runtime options report only displays options that are specified inside the CEEPRMxx parmlib members.

Syntax



xx The two alphanumeric characters that are the suffix of the CEEPRMxx parmlib members to be checked. The **MEMBERS** keyword parameter must always be specified.

data-set-name

The data set name that contains the specified CEEPRMxx parmlib member. The fully qualified data set name must be enclosed in single quotation marks if a TSO/E prefix is not required. The **DSN** and **DSNAME** keyword parameters are optional.

If both DD is allocated and DSN or DSNAME is specified, the CEEPRMCK program uses the DD and the DSN/DSNAME is ignored. DD allocation overrides DSN/DSNAME specification.

If no DD is allocated and no DSN or DSNAME is specified, the CEEPRMCK program uses the default data set SYS1.PARMLIB.

Usage notes

- An input data set must be a fixed record format and a record length of 80.
- To invoke CEEPRMCK by using the documented syntax, SCEECLST must be allocated to a system file (SYSPROC or SYSEXEC). See *z/OS TSO/E REXX Reference* for more information about setting up and using REXX execs.

Return codes

The possible return codes are as follows:

- 0 Successful completion.
- 4 *Keyword parameter* is not a valid option or was specified incorrectly.
- 8 The MEMBERS keyword was not specified or was specified incorrectly.
- 12 DSN and DSNAME keywords cannot be specified at the same time.
- 16 Failed to allocate *parmlib data set*.
- 20 The MEMBERS, DSN, or DSNAME keyword parameter was missing a closing parenthesis.
- 1xxx xxx is the return code from the CEEPRMCC program. See “CEEPRMCC - syntax checking under z/OS batch” on page 36.

Examples

The following example shows how to check the parmlib members CEEPRMMS, CEEPRMPV, and CEEPRMJM which reside in the MEENAK.SYSTEM.PARMLIB data set.

```
CEEPRMCK MEMBERS(MS,PV,JM) DSN('MEENAK.SYSTEM.PARMLIB')
```

Creating region-level runtime option defaults with CEEXOPT

Your site might need to set region-level runtime option defaults that are different from the system-level defaults (if present) or the IBM-supplied defaults. For example, one CICS region (Region A) can be designated to run only AMODE 31 programs, while another region (Region B) runs both AMODE 24 and AMODE 31 programs. This requires Region B to have the ALL31(OFF) option setting while Region A can perform better with the ALL31(ON) option setting. You can accommodate this need by creating separate region-level runtime option load modules for Region A and Region B.

The CEE.SCEESAMP data set contains sample jobs and assembler source files needed to create region-level load modules. In the sample assembler source files, all runtime options are coded with the IBM-supplied default suboption values. When the sample jobs are used to assemble the source files, the CEEXOPT macro is invoked to create the CEEROPT (AMODE31) or CELQROPT (AMODE 64) load modules. The following table summarizes the samples provided in CEE.SCEESAMP:

Table 8. Samples for creating region-level runtime option load modules

Set defaults for	Sample job	Assembler source
Region-level z/OS batch/IMS/LRR	CEEWROPT	CEERDOPT (shown in "Sample invocation of CEEXOPT within the CEERDOPT member")
Region-level CICS	CEEWROPT	CEERCOPT (shown in "Sample invocation of CEEXOPT within the CEERCOPT member" on page 41)
Region-level z/OS batch (AMODE 64)	CEEWQROP	CELQRDOP (shown in "Sample invocation of CEEXOPT within the CELQRDOP member" on page 42)

The CEEWROPT and CEEWQROP jobs do not use SMP/E to create the region-level load modules, so you can run them several times to create several different load modules, each in their own specific library. The load modules can then be included as part of the STEPLIB concatenation. If a CEEROPT or CELQROPT load module is present in a program search order, Language Environment loads and merges the specified options. Any region-level options specified will override the system-level defaults (if present and overridable) and the IBM-supplied defaults. Language Environment does not ship a default CEEROPT or CELQROPT load module.

Sample invocation of CEEXOPT within the CEERDOPT member

```

CEEROPT CSECT
CEEROPT AMODE ANY
CEEROPT RMODE ANY
CEEXOPT ABPERC=((NONE),OVR), X
        ABTERMENC=((ABEND),OVR), X
        AIXBLD=((OFF),OVR), X
        ALL31=((ON),OVR), X
        ANYHEAP=((16K,8K,ANYWHERE,FREE),OVR), X
        BELOWHEAP=((8K,4K,FREE),OVR), X
        CBOPTS=((ON),OVR), X
        CBLPSHPOP=((ON),OVR), X
        CBLQDA=((OFF),OVR), X
        CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR), X
        CHECK=((ON),OVR), X
        COUNTRY=((US),OVR), X
        DEBUG=((OFF),OVR), X
        DEPTHCONDLMT=((10),OVR), X
        DYNDDUMP=((*USERID,NODYNAMIC,TDUMP),OVR), X
        ENVAR=(('),OVR), X
        ERRCOUNT=((0),OVR), X
        ERRUNIT=((6),OVR), X
        FILEHIST=((ON),OVR), X
        FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR), X

```

```

HEAP=((32K,32K,ANYWHERE,KEEP,8K,4K),OVR), X
HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR), X
HEAPOOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048, X
10,0,10,0,10,0,10,0,10,0,10,0,10),OVR), X
INFOMSGFILTER=((OFF,,,),OVR), X
INQPCOPN=((ON),OVR), X
INTERRUPT=((OFF),OVR), X
LIBSTACK=((4K,4K,FREE),OVR), X
MSGFILE=((SYSOUT,FBA,121,0,NOENQ),OVR), X
MSGQ=((15),OVR), X
NATLANG=((ENU),OVR), X
NOAUTOTASK=(OVR), X
NOTEST=((ALL,*,PROMPT,INSPPREF),OVR), X
NOUSRHDLR=((),OVR), X
OCSTATUS=((ON),OVR), X
PC=((OFF),OVR), X
PLITASKCOUNT=((20),OVR), X
POSIX=((OFF),OVR), X
PROFILE=((OFF,''),OVR), X
PRTUNIT=((6),OVR), X
PUNUNIT=((7),OVR), X
RDRUNIT=((5),OVR), X
RECPAD=((OFF),OVR), X
RPTOPTS=((OFF),OVR), X
RPTSTG=((OFF),OVR), X
RTEREUS=((OFF),OVR), X
SIMVRD=((OFF),OVR), X
STACK=((128K,128K,ANYWHERE,KEEP,512K,128K),OVR), X
STORAGE=((NONE,NONE,NONE,0K),OVR), X
TERMTHDACT=((TRACE,,96),OVR), X
THREADHEAP=((4K,4K,ANYWHERE,KEEP),OVR), X
THREADSTACK=((OFF,4K,4K,ANYWHERE,KEEP,128K,128K),OVR), X
TRACE=((OFF,4K,DUMP,LE=0),OVR), X
TRAP=((ON,SPIE),OVR), X
UPSI=((00000000),OVR), X
VCTRSAVE=((OFF),OVR), X
XUFLOW=((AUTO),OVR)

```

END

Sample invocation of CEEXOPT within the CEERCOPT member

```

CEERCOPT CSECT
CEERCOPT AMODE ANY
CEERCOPT RMODE ANY
CEEXOPT ABPERC=((NONE),OVR), X
ABTERMENC=((ABEND),OVR), X
AIXBLD=((OFF),OVR), X
ALL31=((ON),OVR), X
ANYHEAP=((4K,4080,ANYWHERE,FREE),OVR), X
BELOWHEAP=((4K,4080,FREE),OVR), X
CBLOPTS=((ON),OVR), X
CBLPSHPOP=((ON),OVR), X
CBLQDA=((OFF),OVR), X
CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR), X
CHECK=((ON),OVR), X
COUNTRY=((US),OVR), X
DEBUG=((OFF),OVR), X
DEPTHCONDLMT=((10),OVR), X
DYNDUMP=((*USERID,NODYNAMIC,TDUMP),OVR), X
ENVAR=(('),OVR), X
ERRCOUNT=((0),OVR), X
ERRUNIT=((6),OVR), X
FILEHIST=((ON),OVR), X
FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR), X
HEAP=((4K,4080,ANYWHERE,KEEP,4K,4080),OVR), X

```

CEEXOPT

```
|
|                                     HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR),          X
|                                     HEAPPOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048, X
|                                     10,0,10,0,10,0,10,0,10,0,10,0,10),OVR),          X
|                                     INFOMSGFILTER=((OFF,,,,),OVR),                X
|                                     INQPCOPN=((ON),OVR),                          X
|                                     INTERRUPT=((OFF),OVR),                        X
|                                     LIBSTACK=((32,4080,FREE),OVR),                 X
|                                     MSGFILE=((SYSOUT,FBA,121,0,NOENQ),OVR),        X
|                                     MSGQ=((15),OVR),                              X
|                                     NATLANG=((ENU),OVR),                          X
|                                     NOAUTOTASK=(OVR),                             X
|                                     NOTEST=((ALL,*,PROMPT,INSPREF),OVR),          X
|                                     NOUSRHDLR=((),OVR),                           X
|                                     OCSTATUS=((ON),OVR),                          X
|                                     PC=((OFF),OVR),                               X
|                                     PLITASKCOUNT=((20),OVR),                     X
|                                     POSIX=((OFF),OVR),                            X
|                                     PROFILE=((OFF,''),OVR),                       X
|                                     PRTUNIT=((6),OVR),                             X
|                                     PUNUNIT=((7),OVR),                             X
|                                     RDRUNIT=((5),OVR),                             X
|                                     RECPAD=((OFF),OVR),                            X
|                                     RPTOPTS=((OFF),OVR),                          X
|                                     RPTSTG=((OFF),OVR),                           X
|                                     RTEREUS=((OFF),OVR),                           X
|                                     SIMVRD=((OFF),OVR),                             X
|                                     STACK=((4K,4080,ANYWHERE,KEEP,4K,4080),OVR),    X
|                                     STORAGE=((NONE,NONE,NONE,0K),OVR),            X
|                                     TERMTHDACT=((TRACE,CESE,96),OVR),              X
|                                     THREADHEAP=((4K,4080,ANYWHERE,KEEP),OVR),      X
|                                     THREADSTACK=((OFF,4K,4080,ANYWHERE,KEEP,4K,4080),OVR), X
|                                     TRACE=((OFF,4K,DUMP,LE=0),OVR),                 X
|                                     TRAP=((ON,SPIE),OVR),                           X
|                                     UPSI=((00000000),OVR),                          X
|                                     VCTRSAVE=((OFF),OVR),                           X
|                                     XUFLOW=((AUTO),OVR)                             X
|
|                                     END
```

Sample invocation of CEEXOPT within the CELQRDOP member

```
|
| CELQROPT CSECT
| CELQROPT AMODE 64
| CELQROPT RMODE ANY
|                                     CEEXOPT CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR), X
|                                     DYNDUMP=((*USERID,NODYNAMIC,TDUMP),OVR),          X
|                                     ENVAR=(('),OVR),                                X
|                                     FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR),            X
|                                     HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR),      X
|                                     HEAPPOLS=((OFF,8,10,32,10,128,10,256,10,1024,10, X
|                                     2048,10,0,10,0,10,0,10,0,10,0,10,0,10),OVR),    X
|                                     HEAPPOLS64=((OFF,8,4000,32,2000,128,700,256,350, X
|                                     1024,100,2048,50,3072,50,4096,50,8192,25,16384,10, X
|                                     32768,5,65536,5),OVR),                          X
|                                     HEAP64=((1M,1M,KEEP,32K,32K,KEEP,4K,4K,FREE),OVR), X
|                                     INFOMSGFILTER=((OFF,,,,),OVR),                  X
|                                     IOHEAP64=((1M,1M,FREE,12K,8K,FREE,4K,4K,FREE),OVR), X
|                                     LIBHEAP64=((1M,1M,FREE,16K,8K,FREE,8K,4K,FREE),OVR), X
|                                     NATLANG=((ENU),OVR),                            X
|                                     NOTEST=((ALL,*,PROMPT,INSPREF),OVR),            X
|                                     POSIX=((OFF),OVR),                             X
|                                     PROFILE=((OFF,''),OVR),                         X
|                                     RPTOPTS=((OFF),OVR),                           X
|                                     RPTSTG=((OFF),OVR),                             X
|                                     STACK64=((1M,1M,128M),OVR),                     X
|                                     STORAGE=((NONE,NONE,NONE,),OVR),                X
```

```

|                                     THREADSTACK64=((OFF,1M,1M,128M),OVR),           X
|                                     TERMTHDACT=((TRACE,,96),OVR),                 X
|                                     TRACE=((OFF,,DUMP,LE=0),OVR),                 X
|                                     TRAP=((ON,SPIE),OVR)
|
|                                     END

```

CEEXOPT invocation for CEEROPT (AMODE 31)

To invoke CEEXOPT and create the CEEROPT load module, do the following:

1. Copy member CEERDOPT (non-CICS) or CEERCOPT (CICS) from CEE.SCEESAMP into CEEWROPT in place of the comment lines following the SYSIN DD statement.
2. Change the parameters on the CEEXOPT macro statement to reflect the values you have chosen for this region-level load module.
3. Code only the options that you want to change. Options omitted remain same as the system-level defaults (if present) or the IBM-supplied defaults.
4. Change DSNAME=YOURLIB in the SYSLMOD DD statement to the name of the partitioned data set into which you want your CEEROPT load module to be link-edited. This data set does not need to be APF authorized.

Note: If you have a CEEROPT load module in the specified data set, it will be replaced by the new version.

5. Check the SYSLIB DD statement to ensure the data set names are correct.

CEEWROPT should run with a condition code of 0.

Note: CICS supports XPLINK programs in a CICS environment. The CICS region defaults are not used for these programs. However, if the CEEPRMxx parmlib member keyword CEEROPT is set to ALL, the XPLINK programs can use the region defaults that can be located within the MVS search order.

CEEXOPT invocation for CELQROPT (AMODE 64)

To invoke CEEXOPT and create the CELQROPT load module, follow these steps:

1. Copy member CELQRDOP from CEE.SCEESAMP into CEEWQROP in place of the comment lines following the SYSIN DD statement.
2. Change the parameters on the CEEXOPT macro statement to reflect the values that you chose for this region-level load module.
3. Code only the options that you want to change. Options that you omit from CELQROPT remain the same as the system-level defaults (if present) or the IBM-supplied defaults.
4. Change DSNAME=YOURLIB in the SYSLMOD DD statement to the name of the partitioned data set into which you want your CELQROPT load module to be link-edited.

Note: If you have a CELQROPT module in the specified data set, it will be replaced by the new version.

5. Check the SYSLIB DD statement to ensure that the data set names are correct.

CEEWQROP should run with a condition code of 0.

CEEXOPT coding guidelines for CEEROPT and CELQROPT

Be aware of the following coding guidelines for the CEEXOPT macro:

CEEXOPT

- A continuation character (X in the source) must be present in column 72 on each line of the CEEXOPT invocation except the last line. The continuation line must start in column 16. You can break the coding after any comma.
- Options and suboptions must be specified in uppercase. Only suboptions that are strings can be specified in mixed case or lowercase. For example, both MSGFILE=(SYSOUT) and MSGFILE=(sysout) are acceptable.
- A comma must end each option except for the final option. If the comma is omitted, everything following the option is treated as a comment.
- If one of the string suboptions contains a special character, such as 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 adjacent apostrophes or adjacent quotation marks.
To get a single apostrophe (') or a single ampersand (&) within a string, two instances of the character must be specified. The pair is counted as only one character in determining if the maximum allowable string length has been exceeded, and in setting the effective length of the string.
- Avoid unmatched apostrophes in any string. The error cannot be captured within CEEXOPT itself; instead, the assembler produces a message such as:
IEV03 *** ERROR *** NO ENDING APOSTROPHE

which bears no particular relationship to the suboption in which the apostrophe was omitted. Furthermore, none of the options are parsed properly if this mistake is made.

- Macro instruction operands cannot be longer than 1024 characters. If the number of characters to the right of the equal sign is greater than 1024 for any keyword parameter in the CEEXOPT invocation, a return code of 12 is produced for the assembly, and the options are not parsed properly.
- You can completely omit the specification of any runtime option. Options not specified are ignored at the time Language Environment merges the options.
- Any options specified must be designated as overridable (OVR) or nonoverrideable (NONOVR).

Performance considerations

For optimal performance when using CEEROPT or CELQROPT, code only those options that you want to change. This enhances performance by minimizing the number of options that Language Environment must merge at runtime. Options and suboptions that are to remain the same as the defaults do not need to be repeated. For example, if the only change you want to make is to define STACK with an initial value of 64K and an increment of 64K, include only that runtime option, as shown in the following example:

```
CEEROPT CSECT
CEEROPT AMODE ANY
CEEROPT RMODE ANY
          CEEXOPT STACK=((64K,64K,ANYWHERE,KEEP,512K,128K),OVR)
          END
```

Chapter 6. Language Environment runtime options

This topic includes descriptions of the Language Environment runtime options. Where noted, some of the runtime options might be used only by a specific program or by specific AMODE applications only.

For a table that maps Language Environment runtime options to HLL runtime options to help you plan your customization, see *z/OS Language Environment Runtime Application Migration Guide*.

Note: CICS TS 3.1 and higher supports XPLINK programs in a CICS environment. The default values for these programs are the non-CICS default values.

Cobol compatibility

Restriction: COBOL is not supported for AMODE 64 applications.

VS COBOL II supported an order of runtime options and program options that is the reverse of that of Language Environment: program arguments precede runtime options in COBOL. To ensure compatibility with COBOL, Language Environment provides the runtime option CBLOPTS, which specifies whether runtime options or program arguments are first in the character parameter.

For example:

```
CBLOPTS=OFF
//GO EXEC PGM=PROGRAM1,PARM='AIXBLD/'

CBLOPTS=ON
//GO EXEC PGM=PROGRAM1,PARM=' /AIXBLD'
```

Runtime options

The runtime options that can be modified at the system or region level are described in the format specific to CEEPRMxx, CEEROPT and CELQROPT. You do not have to specify all of the options, and abbreviations are not permitted.

IBM-supplied default keywords are indicated for planning information only and appear above the main path or options path in the syntax diagrams. In the parameter list, IBM-supplied default choices are underlined.

Some of these runtime options descriptions refer to the severity of conditions. The values that can occur as condition token severity codes, and their meanings, are as follows:

- 0 An informational message (or, if the entire token is zero, no information).
- 1 An attention message. Service completed, probably correctly.
- 2 An error message. Correction attempted. Service completed, perhaps incorrectly.
- 3 A severe error message. Service not completed.
- 4 A critical error message. Service not completed and condition signaled. A critical error is a condition that jeopardizes the environment. If a critical

error occurs during a Language Environment callable service, it is always signaled to the condition manager instead of being returned synchronously to the caller.

For a complete description of all Language Environment runtime options, see *z/OS Language Environment Programming Reference*.

ABPERC

Derivation: ABnormal PERColation

ABPERC percolates an abend whose code you specify. This provides Language Environment condition handling semantics for all abends except the one specified.

Restriction: TRAP(ON) must be in effect.

When you run with ABPERC and encounter the specified abend:

- User condition handlers are not enabled.
- In z/OS UNIX, POSIX signal handling semantics are not enabled for the abend.
- No storage report or runtime options report is generated.
- No Language Environment messages or Language Environment dump output is generated.
- The assembler user exit is not driven for enclave termination.
- The abnormal termination exit (if there is one) is not driven.
- Files opened by HLLs are not closed by Language Environment, so records might be lost.
- Resources acquired by Language Environment are not freed.
- The debug tool is not notified of the error.

Tip: You can also use the CEEBXITA assembler user exit to specify a list of abend codes for Language Environment to percolate. For more information about CEEBXITA, see *z/OS Language Environment Programming Guide*

Non-CICS default

ABPERC=((NONE),OVR)

CICS default

ABPERC is ignored under CICS.

Syntax

```
▶▶ ABPERC == ( ( [NONE] abcode ) , [OVR] NONOVR ) ▶▶
```

NONE

Specifies that all abends are handled according to Language Environment condition handling semantics.

abcode

Specifies the code number of the abend to percolate. The *abcode* can be specified as:

Shhh A system abend code, where *hhh* is the hex system abend code

Udddd A user abend code, where *dddd* is a decimal user-issued abend code

Any 4-character string can also be used as an *abcode*.

You can identify only one abend code with this option. However, an abend U0000 is interpreted in the same way as S000.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

ABPERC percolates an abend regardless of the thread in which it occurs.

Usage notes

Language Environment ignores ABPERC(OCx). In this instance, no abend is percolated, and Language Environment condition handling semantics are in effect.

For more information

For more information about the assembler user exit (CEEEXITA), see *z/OS Language Environment Programming Guide*.

ABTERMENC

Derivation: ABnormal TERMination of the ENClave

ABTERMENC sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater. TRAP(ON) must be in effect for ABTERMENC to have an effect.

Non-CICS default

ABTERMENC=((ABEND),OVR)

CICS default

ABTERMENC=((ABEND),OVR)

Syntax

```

  >> ABTERMENC = ( ( ABEND
                    RETCODE ) , ( OVR
                                 NONOVR ) ) >>
  
```

ABEND

Specifies that Language Environment issues an abend to end the enclave

ABTERMENC

regardless of the setting of the CEEAUE_ABND flag by the assembler user exit. However, the setting of the CEEAUE_ABND flag affects the abend processing, as follows:

When CEEAUE_ABND is set to OFF, the following occurs:

- **Abend code:** Language Environment sets an abend code value that depends on the type of unhandled condition.
- **Reason code:** Language Environment sets a reason code value that depends on the type of unhandled condition.
- **Abend dump attribute:** Language Environment does not request a system dump.
- **Abend task/step attribute (on z/OS):** An abend is issued to terminate the task.

When CEEAUE_ABND is set to ON, Language Environment uses values set by the assembler user exit to determine abend processing:

- **Abend code:** Value of the CEEAUE_RETC parameter of the assembler user exit.
- **Reason code:** Value of the CEEAUE_RSNC parameter of the assembler user exit.
- **Abend dump attribute:** Language Environment requests a system dump only if the assembler user exit sets CEEAUE_DUMP to ON. The system abend dump goes to the system abend ddname with the file name you define in your JCL. The file name is the name defined in the DD card.
- **Abend task/step attribute (on z/OS):** If the assembler user exit sets CEEAUE_STEPS to ON, Language Environment issues an abend to terminate the step. Otherwise, Language Environment issues an abend to terminate the task.

RETCODE

Specifies that the enclave terminates with a nonzero return code.

However, the assembler user exit can modify this behavior as follows:

- If the assembler user exit does not set the CEEAUE_ABND flag to ON during enclave termination, Language Environment returns to its caller with a return code and a reason code.
- If the assembler user exit sets the CEEAUE_ABND flag to ON during enclave termination, Language Environment issues an abend to terminate the enclave. Language Environment sets the abend and reason code for the abend to equal the values of assembler user exit parameters, as follows:
 - **Abend code:** Value of the CEEAUE_RETC parameter of the assembler user exit. If the assembler user exit does not modify the CEEAUE_RETC value, Language Environment sets an abend code that maps to the severity of the condition and to the user return code.
 - **Reason code:** Value of the CEEAUE_RSNC parameter of the assembler user exit.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

In a multithreaded application with ABEND set for ABTERMENC only the main(IPT) thread will be ABENDED and the application terminated, regardless of which thread experienced the unhandled condition. All other threads (the NON-IPT threads) will be terminated normally, including the offending thread, if it is a NON-IPT thread.

Usage notes

When running with IMS, the ABTERMENC(ABEND) setting ensures that IMS transactions are rolled back if errors occur in an application that is written in another Language Environment-enabled language; an abend causes IMS to roll back any database updates.

When running a batch job, the ABTERMENC(ABEND) setting ensures that a job step will abend if errors occur in an application that is written in another language.

For more information

- For information about return code calculation CEEAUE_RETC, CEEAUE_ABND, and assembler user exit CEEBXTA processing, see *z/OS Language Environment Programming Guide*.
- For more information about abend codes and a list of abend code values see *z/OS Language Environment Programming Guide*.

AIXBLD (COBOL only)

Derivation: Alternate IndeX BuiLD

AIXBLD invokes the access method services (AMS) for VSAM indexed and relative data sets (KSDS and RRDS) to complete the file and index definition procedures for COBOL programs.

AIXBLD conforms to the ANSI 1985 COBOL standard.

Non-CICS default

AIXBLD=((OFF),OVR)

CICS default

AIXBLD is ignored under CICS.

Syntax

► AIXBLD = ((OFF | ON) , (OVR | NONOVR)) ◀

OFF

Does not invoke the access method services for VSAM indexed and relative data sets.

- ON** Invokes the access method services for VSAM indexed and relative data sets. AIXBLD can be abbreviated AIX®.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

If you also specify the MSGFILE runtime option, the access method services messages are directed to the MSGFILE *ddname* or to the default SYSOUT.

Usage notes

The only valid abbreviations for AIXBLD and NOAIXBLD are AIX and NOAIX, respectively.

Performance considerations

Running your program under AIXBLD requires more storage, which can degrade performance. Therefore, use AIXBLD only during application development to build alternate indexes. Use NOAIXBLD when you have already defined your VSAM data sets.

ALL31

Derivation: ALL AMODE 31

ALL31 specifies whether an application can run entirely in AMODE 31 or whether the application has one or more AMODE 24 routines.

This option does not implicitly alter storage, in particular storage managed by the STACK and HEAP runtime options. However, you must be aware of your application's requirements for stack and heap storage, because such storage can potentially be allocated above the line while running in AMODE 24.

It is recommended that ALL31 have the same setting for all enclaves in a process. Language Environment does not support the invocation of a nested enclave requiring ALL31(OFF) from an enclave running with ALL31(ON) in non-CICS environments.

In a multithread environment, Language Environment invokes all start routines, which are specified in a Language Environment `pthread_create()` function call, in AMODE 31. However, for PL/I MTF applications, Language Environment provides AMODE switching. Thus, the first routine of a task can be in AMODE 24.

Non-CICS default:

ALL31=((ON),OVR)

CICS default

ALL31=((ON),OVR)

Syntax

```

  ┌───┐ ┌───┐ ┌───┐ ┌───┐
  │   │ │   │ │   │ │   │
  └───┘ └───┘ └───┘ └───┘
  ALL31(ON|OFF), (OVR|NONOVR)
  
```

ON Indicates that no user routines of a Language Environment application are AMODE 24.

With ALL31(ON) specified:

- AMODE switching across calls to Language Environment common runtime routines is minimized. For example, no AMODE switching is performed on calls to Language Environment callable services.

OFF

Indicates that one or more routines of a Language Environment application are AMODE 24.

With ALL31(OFF) specified:

- AMODE switching across calls to Language Environment common runtime routines is performed. For example, AMODE switching is performed on calls to Language Environment callable services.
- In COBOL, EXTERNAL data is allocated in storage below the 16-MB line.

If you use the setting ALL31(OFF), you must also use the setting STACK(„BELOW,,“). AMODE 24 routines require that stack storage is below the 16-MB line.

If you use the setting ALL31(OFF), Language Environment preallocates BELOWHEAP instead of ANYHEAP storage.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

With ALL31(ON), Language Environment allocates storage for the common anchor area (CAA) and other control blocks in unrestricted storage.

z/OS UNIX considerations

The ALL31 option applies to the enclave.

Usage notes

When an application is running in an XPLINK environment (that is, either the XPLINK(ON) runtime option was specified, or the initial program contained at least one XPLINK-compiled part), the ALL31 runtime option will be forced to ON. No AMODE 24 routines are allowed in an enclave that uses XPLINK. No message will be issued to indicate this action. In this case, if a Language Environment runtime options report is generated using the RPTOPTS runtime option, the ALL31 option will be reported as "Override" under the LAST WHERE SET column.

COBOL considerations

You must specify ALL31(OFF) if your applications contain one of the following programs:

- A VS COBOL II NORES program
- An OS/VS COBOL program (non-CICS program)
- An AMODE 24 program

Fortran considerations

Use ALL31(ON) if all of the compile units in the enclave have been compiled with VS FORTRAN Version 1 or Version 2 and there are no requirements for 24-bit addressing mode. Otherwise, use ALL31(OFF).

Performance considerations

If your application consists entirely of AMODE 31 routines, it might run faster and use less below-the-line storage with ALL31(ON) than with ALL31(OFF), since mode switching code is not required.

For more information

- See "STACK" on page 113 for information about the STACK runtime option.

ANYHEAP

ANYHEAP controls the allocation of library heap storage that is not restricted to a location below the 16-MB line.

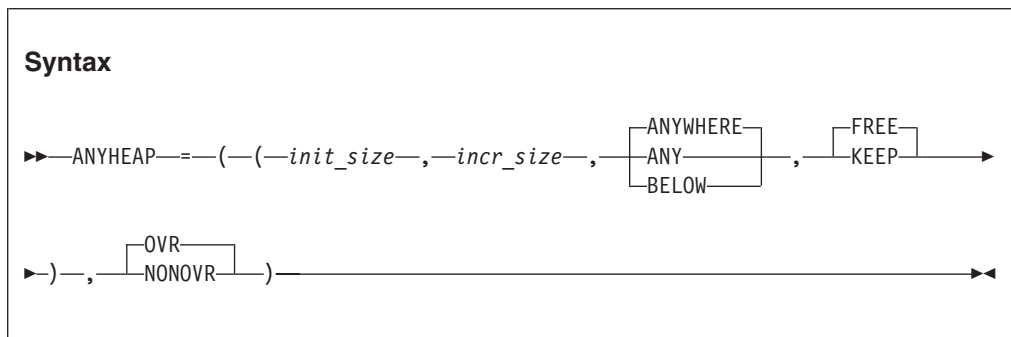
The ANYHEAP option is always in effect. If you do not specify ANYHEAP or if you specify ANYHEAP(0), Language Environment allocates the value of 16K when a call is made to get heap storage.

Non-CICS default

ANYHEAP=((16K,8K,ANYWHERE,FREE),OVR)

CICS default

ANYHEAP=((4K,4080,ANYWHERE,FREE),OVR)



init_size

Determines the minimum initial size of the anywhere heap storage. This value can be specified as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr_size

Determines the minimum size of any subsequent increment to the anywhere

heap area, and is specified in *n*, *nK*, or *nM* bytes of storage. This value is rounded up to the nearest multiple of 8 bytes.

ANYWHERE | ANY

Specifies that heap storage can be allocated anywhere in storage. If there is no storage available above the line, storage is acquired below the 16-MB line.

The only valid abbreviation for ANYWHERE is ANY.

BELOW

Specifies that heap storage must be allocated below the 16-MB line in storage that is accessible to 24-bit addressing.

FREE

Specifies that storage allocated to ANYHEAP increments is released when the last of the storage is freed.

KEEP

Specifies that storage allocated to ANYHEAP increments is **not** released when the last of the storage is freed.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

Both the initial size and the increment size are rounded up to the nearest multiple of 8 bytes. If ANYHEAP(0) is specified, the initial HEAP is obtained on the first use and will be based on the increment size. The maximum initial and increment size for ANYHEAP under CICS is 1 gigabyte (1024 MB).

The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16-byte CICS storage check zone. Without this accommodation, an extra page of storage is allocated (only when the storage allocation is below the 16MB line).

z/OS UNIX considerations

The ANYHEAP option applies to the enclave.

Performance considerations

The ANYHEAP option improves performance when you specify values that minimize the number of times the operating system allocates storage. The RPTSTG runtime generates a report of the storage the application uses while running; you can use the report numbers to help determine what values to specify.

For more information

- See *z/OS Language Environment Programming Guide* for more information about Language Environment heap storage.
- See “RPTSTG” on page 108 for more information about the RPTSTG runtime.
- For more information about heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.

AUTOTASK | NOAUTOTASK (Fortran only)

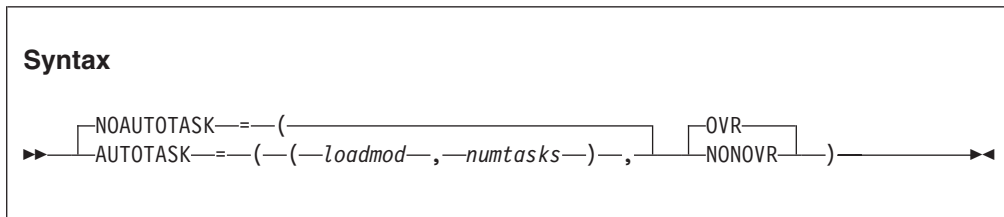
AUTOTASK specifies whether Fortran Multitasking Facility is to be used by your program and the number of tasks that are allowed to be active.

Non-CICS default

NOAUTOTASK=(OVR)

CICS default

AUTOTASK is ignored under CICS.



NOAUTOTASK

Disables the MTF and nullifies the effects of previous specifications of AUTOTASK parameters.

loadmod

The name of the load module that contains the concurrent subroutines that run in the subtasks created by MTF.

numtasks

The number of subtasks created by MTF. This value can range from 1 through 99.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

BELOWHEAP

BELOWHEAP controls the allocation of library heap storage that must be located below the 16MB line. The heap controlled by BELOWHEAP is intended for items such as control blocks used for I/O.

Non-CICS default

BELOWHEAP=((8K,4K,FREE),OVR)

CICS default

BELOWHEAP=((4K,4080,FREE),OVR)

Syntax

```

▶▶ BELOWHEAP(—(—(—init_size—,—incr_size—,—FREEKEEP—))—,—OVRNONOVR—)

```

init_size

Determines the minimum initial size of the below heap storage. This value can be specified as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr_size

Determines the minimum size of any subsequent increment to the area below the 16MB line, and is specified in *n*, *nK*, or *nM* bytes of storage. This value is rounded up to the nearest multiple of 8 bytes.

FREE

Specifies that storage allocated to BELOWHEAP increments is released when the last of the storage is freed.

KEEP

Specifies that storage allocated to BELOWHEAP increments is **not** released when the last of the storage is freed.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16 bytes CICS storage check zone. Without this accommodation, an extra page of storage is allocated (only when the storage allocation is below the 16MB line).

z/OS UNIX considerations

The BELOWHEAP option applies to the enclave.

Usage notes

Both the initial size and the increment size are rounded to the nearest multiple of 8 bytes. If you specify BELOWHEAP(0), the initial BELOWHEAP is obtained on the first use and will be the increment size.

Performance considerations

BELOWHEAP improves performance when you specify values that minimize the number of times that the operating system allocates storage. The RPTSTG run-time option generates a report of storage your application uses while running. You can use its numbers to help determine what values to specify.

For more information

- See *z/OS Language Environment Programming Guide* for more information about Language Environment heap storage.
- See “RPTSTG” on page 108 for more information about the RPTSTG runtime option.
- For more information about tuning your application with storage report numbers, see *z/OS Language Environment Programming Guide*.

CBLOPTS (COBOL only)

Derivation: COBOL OPTIONS

CBLOPTS specifies the format of the parameter string on application invocation when the main program is COBOL. CBLOPTS determines whether run-time options or program arguments appear first in the parameter string.

You can only specify this option at the system level, region level, or in CEEUOPT.

When you specify the ON suboption, the runtime options and program arguments specified in the JCL or on the command line are honored in the following order, which is the reverse of that usually honored by Language Environment:
program arguments/runtime options

CBLOPTS(ON) allows the existing COBOL format of the invocation character string to continue working (user parameters followed by runtime options). CBLOPTS(ON) is valid only for applications whose main program is COBOL.

Non-CICS default

CBLOPTS=((ON),OVR)

CICS default

CBLOPTS is ignored under CICS.

Syntax

►►—CBLOPTS—==(—(—(—ON—OFF—)—),—OVR—NONOVR—)—►►

ON Specifies that program arguments appear first in the parameter string.

OFF

Specifies that runtime options appear first in the parameter string.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

If the string contains only runtime options that are invalid, the entire string is interpreted as a program argument. For example, if you pass the string

11/16/1967, 1967 is interpreted as an invalid runtime option. Since there are no other runtime options, the entire string is interpreted as a program argument.

CBLPSHPOP (COBOL only)

Derivation: COBOL PUSH POP

CBLPSHPOP controls whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL subroutine is called.

Specify CBLPSHPOP(ON) to avoid compatibility problems when calling COBOL subroutines that contain CICS CONDITION, AID, or ABEND condition handling commands.

You can set the CBLPSHPOP run-time option on an enclave basis using CEEUOPT.

CBLPSHPOP is ignored in non-CICS environments.

Non-CICS default

n/a

CICS default

CBLPSHPOP=((ON),OVR)

Syntax

```

▶▶ CBLPSHPOP=( ( ( ON
                OFF
                ) ) , ( OVR
                    NONOVR
                    ) )

```

ON Automatically issues the following when a COBOL subroutine is called:

- An EXEC CICS PUSH HANDLE command as part of the routine initialization.
- An EXEC CICS POP HANDLE command as part of the routine termination.

OFF

Does not issue CICS PUSH HANDLE and CICS POP HANDLE commands on a call to a COBOL subroutine.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

If your application calls COBOL subroutines under CICS, performance is better with CBLPSHPOP(OFF) than with CBLPSHPOP(ON).

For more information

For more information about CEEUOPT, see *z/OS Language Environment Programming Guide*.

CBLQDA (COBOL Only)

Derivation: COBOL QSAM Dynamic Allocation

CBLQDA controls COBOL QSAM dynamic allocation on an OPEN statement.

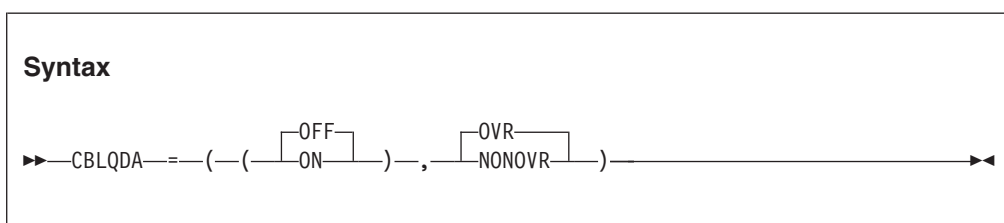
CBLQDA does not affect dynamic storage allocation for the message file specified in MSGFILE or the Language Environment formatted dump file (CEEDUMP).

Non-CICS default:

CBLQDA=((OFF),OVR)

CICS default:

CBLQDA is ignored under CICS.



OFF

Specifies that COBOL QSAM dynamic allocation is not permitted.

ON

Specifies that COBOL QSAM dynamic allocation is permitted. ON conforms to the 1985 COBOL Standard.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

Using CBLQDA(OFF) under z/OS prevents a temporary data set from being created in case there is a misspelling in your JCL. If you specify CBLQDA(ON) and have a misspelling in your JCL, Language Environment creates a temporary file, writes to it, and then z/OS deletes it. You receive a return code of 0 but no output.

CEEDUMP

Derivation: Common Execution Environment DUMP

The CEEDUMP runtime option is used to specify options to control the processing of the Language Environment dump report CEEDUMP.

Non-CICS default

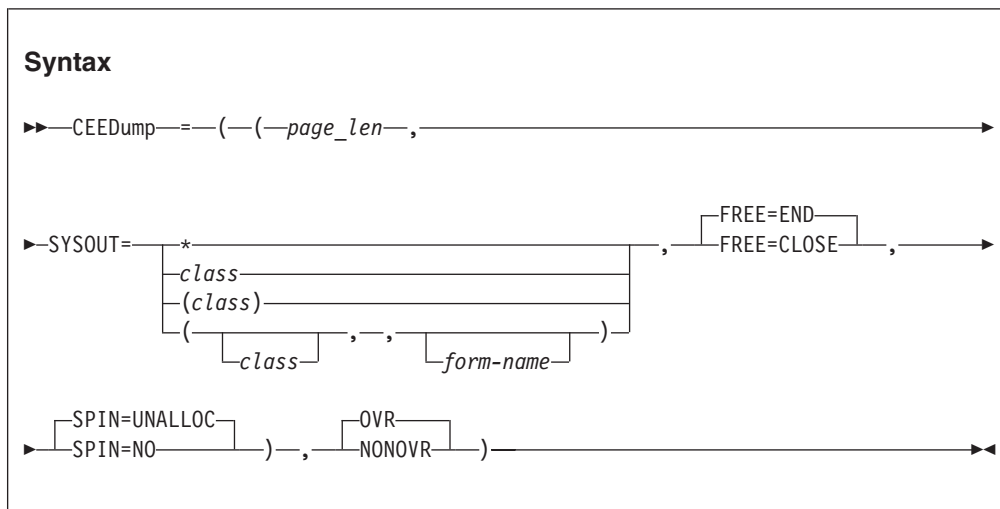
CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR)

CICS default

CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR)

AMODE 64 default

CEEDUMP=((60,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR)

**page_len**

Specifies the number of lines that a CEEDUMP report contains on each page. The number specified by *page_len* must be 0 or a whole number greater than 9. A value of 0 indicates that the dump report contains no page breaks. The default is 60.

The maximum length of *page_len* is limited to 9 digits.

SYSOUT=

Specifies SYSOUT attributes for a dynamically allocated CEEDUMP DD. SYSOUT has three possible parameters, of which two can be specified.

class

Specifies a value that is one character in length. Valid values are A through Z, 0 through 9, and *. A SYSOUT *class* must not be specified inside quotation marks.

- If *class* is not specified, it defaults to * for the dynamically allocated CEEDUMP.
- If dynamic allocation for the specified SYSOUT class fails, SYSOUT=* is set and message CEE3785I is issued.

writer-name

This parameter is not supported and must be omitted.

form-name

Provides a name assigned to an output form for dynamically allocated CEEDUMP DD. *form-name* is made up of 1-4 alphanumeric or national (\$,#,@) characters according to JCL rules. If you want to allow separation of CEEDUMP output from other SYSOUT output for the same class in the JES spool, specify a form in addition to a class for a dynamically allocated CEEDUMP.

FREE=

Specifies that dynamically allocated CEEDUMPs have one of the following JCL DD attributes:

END

The FREE=END DD attribute requests that the system unallocate the data set at the end of the last step that references the data set. This is the default value for this suboption.

CEEDUMP

CLOSE

The FREE=CLOSE DD attribute requests that the system unallocate the data set when it is closed. Code the FREE=CLOSE suboption along with SYSOUT=*class* to make CEEDUMP a spinning data set.

SPIN=

Specifies that dynamically allocated CEEDUMPs have one of the following JCL DD attributes:

UNALLOC

The SPIN=UNALLOC DD attribute indicates that the system needs to make the SYSOUT data set available for processing immediately when it is unallocated. This is the default value for this suboption.

NO The SPIN=NO DD attribute indicates that the system needs to make the SYSOUT data set available for processing as a part of the output at the end of the job, regardless of when the data set is unallocated.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

The SYSOUT=, FREE=, and SPIN= suboptions do not affect the CEEDUMP report taken under CICS

z/OS UNIX considerations

The SYSOUT=, FREE= and SPIN= suboptions do not affect the CEEDUMP report taken in a z/OS UNIX file system.

Usage notes

- If a CEEDUMP DD card is explicitly coded in a job step, Language Environment ignores any SYSOUT *class*, *form-name*, FREE, or SPIN specified in the CEEDUMP runtime.
- The SYSOUT=*class* suboption is overridden by _CEE_DMPTARG when this environment variable is used at the same time to indicate the SYSOUT class.
- The *page_len* suboption is overridden by the CEE3DMP PAGESIZE option. For more information about CEE3DMP, see *z/OS Language Environment Programming Reference*.
- Language Environment supports the use of a CEEDUMP DDNAME dynamically allocated with the XTIO, UCB nocapture, or DSAB-above-the-line options specified in the SVC99 parameters (S99TIOEX, S99ACUCB, S99DSABA flags).

z/OS UNIX considerations

- CEEDUMP=((60,SYSOUT=(C),FREE=END,SPIN=UNALLOC),OVR)
The example changes the default CEEDUMP settings so that the dynamically allocated CEEDUMP output will be sent to sysout class 'C'.
- CEEDUMP=((0,SYSOUT=*,FREE=END,SPIN=UNALLOC),OVR)
The example changes the default CEEDUMP CEEDOPT settings so that the CEEDUMP *pagelen* is 0. This suppresses all page breaks in all the CEEDUMP reports.

CHECK (COBOL only)

This option applies to Enterprise COBOL V4R2 and earlier releases. Starting with Enterprise COBOL V5R1, if the compile time option SSRANGE is specified, range checks are generated by the compiler and the checks are always executed at runtime.

CHECK flags checking errors within an application. In COBOL, index, subscript, and reference modification ranges are checking errors. COBOL is the only language that uses the CHECK option.

Non-CICS default

CHECK=((ON),OVR)

CICS default

CHECK=((ON),OVR)

Syntax

```

▶▶ CHECK = ( ( ON OFF ) , ( OVR NONOVR ) ) ▶▶

```

ON Specifies that runtime checking is performed.

OFF Specifies that runtime checking is not performed.

OVR Specifies that the option can be overridden.

NONOVR Specifies that the option cannot be overridden.

Usage notes

CHECK(ON) has no effect if NOSSRANGE was in effect at compile time.

Performance considerations

If your COBOL program was compiled with SSRANGE, and you are not testing or debugging an application, performance improves when you specify CHECK(OFF).

COUNTRY

COUNTRY sets the country code, which affects the date and time formats, the currency symbol, the decimal separator, and the thousands separator, based on a specified country. COUNTRY does not change the default settings for the language currency symbol, decimal point, thousands separator, and date and time picture strings set by CEESETL or set locale(). COUNTRY affects only the Language Environment NLS services, not the Language Environment locale callable services.

You can set the country value using the runtime option COUNTRY or the callable service CEE3CTY.

COUNTRY

The COUNTRY setting affects the format of the date and time in the reports generated by the RPTOPTS and RPTSTG runtime options.

Non-CICS default

COUNTRY=((US),OVR) with US signifying the United States.

CICS default

COUNTRY=((US),OVR) with US signifying the United States.

Syntax

```
► COUNTRY = - ( - country_code - ) - , [ OVR / NONOVR ]
```

country_code

A 2-character code that indicates to Language Environment the country on which to base the default settings.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

The COUNTRY option sets the initial value for the enclave.

Usage notes

If you specify an unsupported *country_code*, Language Environment accepts the value and issues an informational message. When you specify an unavailable country code, you must provide a message template for that code. CEEUOPT and CEEROPT permit the specification of an unavailable country code, but give a return code of 4 and a warning message.

C/C++ considerations

Language Environment provides locales that are used in Language Environment and C++ to establish default formats for the locale-sensitive functions and locale callable services, such as date and time formatting, sorting, and currency symbols. To change the locale, you can use the `setlocale()` library function or the CEESETL callable service.

The settings of CEESETL or `setlocale()` do not affect the setting of the COUNTRY runtime option. COUNTRY affects only Language Environment NLS and date and time services. `setlocale()` and CEESETL affect only C/C++ locale-sensitive functions and Language Environment locale callable services.

To ensure that all settings are correct for your country, use COUNTRY and either CEESETL or `setlocale()`.

For more information

- For more information about the CEE3CTY and the CEESETL callable services, see *z/OS Language Environment Programming Reference*.

- For more information about `setlocale()`, see *z/OS XL C/C++ Programming Guide*.
- For a list of countries and their codes, see “National language support country codes for Language Environment” on page 271 and *z/OS Language Environment Programming Reference*.

DEBUG (COBOL only)

DEBUG activates the COBOL batch debugging features specified by the USE FOR DEBUGGING declarative.

Non-CICS default

DEBUG=((OFF),OVR)

CICS default

DEBUG=((OFF),OVR)

Syntax

The diagram shows the syntax for the DEBUG declarative: `DEBUG=(((OFF | ON)) , (OVR | NONOVR))`. Brackets indicate that the options OFF and ON are mutually exclusive, and OVR and NONOVR are mutually exclusive. The entire declarative is enclosed in parentheses.

OFF

Suppresses the COBOL batch debugging features.

ON

Activates the COBOL batch debugging features. You must have the WITH DEBUGGING MODE clause in the environment division of your application in order to compile the debugging sections.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Performance considerations

Because DEBUG(ON) gives worse runtime performance than DEBUG(OFF), use it only during application development or debugging.

For more information

For more information about the USE FOR DEBUGGING declarative, see the appropriate version of the COBOL programming guide in the COBOL library at Enterprise COBOL for z/OS library (<http://www-01.ibm.com/support/docview.wss?uid=swg27036733>).

DEPTHCONDLMT

Derivation: DEPTH of nested CONDition LiMiT

DEPTHCONDLMT

DEPTHCONDLMT specifies the extent to which conditions can be nested. Figure 1 illustrates the effect of DEPTHCONDLMT(3) on condition handling. The initial condition and two nested conditions are handled in this example. The third nested condition is not handled.

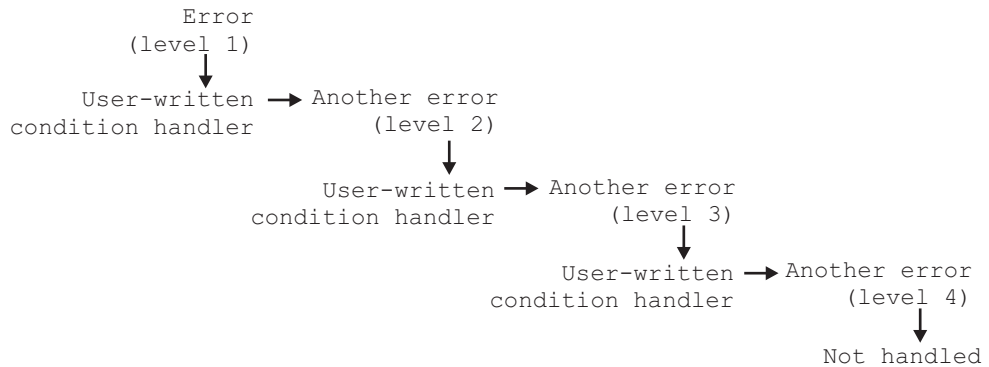


Figure 1. Effect of DEPTHCONDLMT(3) on condition handling

Non-CICS default

DEPTHCONDLMT=((10),OVR)

CICS default

DEPTHCONDLMT=((10),OVR)

Syntax

```
▶▶DEPTHCONDLMT=--(--(limit)--, OVRNONOVR)--▶▶
```

limit

An integer of 0 or greater value. It is the depth of condition handling allowed. An unlimited depth of condition handling is allowed if you specify 0. A 1 value specifies handling of the initial condition, but does not allow handling of nested conditions that occur while handling a condition. With a 5 value, for example, the initial condition and four nested conditions are processed, but there can be no further nesting of conditions.

If the number of nested conditions exceeds the limit, the application terminates with abend U4087.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX consideration

The DEPTHCONDLMT option sets the limit for how many nested synchronous conditions are allowed for a thread. Asynchronous signals do not affect DEPTHCONDLMT.

Usage notes

PL/I considerations

DEPTHCONDLMT(0) provides PL/I compatibility.

PL/I MTF considerations

In a PL/I MTF application, DEPTHCONDLMT sets the limit for how many nested synchronous conditions are allowed for a PL/I task. If the number of nested conditions exceeds the limit, the application terminates abnormally.

For more information

For more information about nested conditions, see *z/OS Language Environment Programming Guide*.

DYNDUMP

Derivation: DYNamic DUMP

The DYNDUMP runtime option provides a way to obtain dynamic dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement.

The dynamic dump is written when:

- Certain types of ABENDs occur. You can select whether a U4039 ABEND or other U40xx ABEND types can cause a dump to be collected.
- The first suboption defines the high-level qualifier of the dynamic dump data set name.
- The second suboption controls when dynamic dumps are taken for U4039 ABENDS.
- The third suboption controls when dynamic dumps are taken for other U40xx ABENDS.

The dump is written to a z/OS data set. It cannot be part of a z/OS UNIX file system.

Non-CICS default

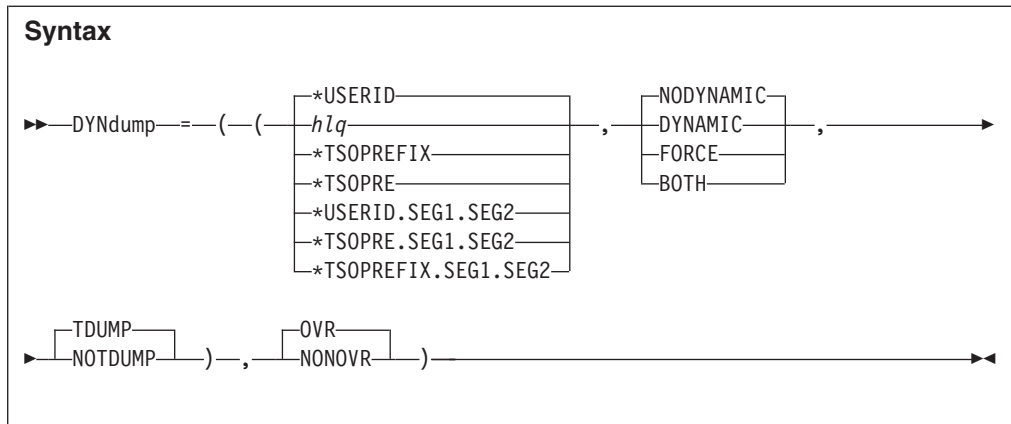
DYNDUMP=((*USERID,NODYNAMIC,TDUMP),OVR)

CICS default

DYNDUMP is ignored.

AMODE 64 default

DYNDUMP=((*USERID,NODYNAMIC,TDUMP),OVR)



hlq

Is a high-level qualifier for the dynamic dump data set to be created. This is concatenated with a time stamp consisting of the Julian day and the time of the dump. The job name or PID can also be part of the name if the combined length of *hlq* and the time stamp is 35 characters or less.

hlq is limited to 26 characters including dot (.) separators. *hlq* allows three keywords:

***USERID**

Tells Language Environment to use the user ID associated with the job step task as the high-level qualifier for the dynamic dump data set.

***TSOPREFIX or *TSOPRE**

Tells Language Environment to use the TSO/E prefix. Each keyword may be followed by additional characters to be used to create the data set name. When appended to the user ID or the TSO prefix, they form the *hlq* used when creating the dump data set.

Restriction: The prefix is only valid in a TSO/E environment. If the prefix is not available, the user ID is used.

The data set name is limited to 44 characters. If the combined length of *hlq* and the time stamp is 35 characters or less, the job name or PID is added to the data set name.

If the system is using multilevel security, the SECLABEL is used as the second qualifier. If *hlq* contains multiple qualifiers, only the first is used, followed by the SECLABEL. The format of the data set name is:

- When the application is not exec()ed and not multilevel security:
hlq.Djjj.Thhmsst.jobname
- When the application is exec()ed and not multilevel security:
hlq.Djjj.Thhmsst.Pppppppp
- When the application is multilevel security and not exec()ed:
hlq.MLS-1evel.Djjj.Thhmsst.jobname
- When the application is both multilevel security and exec()ed:
hlq.MLS-1evel.Djjj.Thhmsst.Pppppppp

For U4039 ABENDS

The following suboptions are used for U4039 ABENDS only:

DYNAMIC

Language Environment creates a dynamic dump automatically when the application has not already specified one of the dump ddnames, (for example, SYSUDUMP).

NODYNAMIC

Language Environment does not create a dynamic dump if no dump DD names are specified.

FORCE

Language Environment always creates a dynamic dump even if other dump DD names have been specified. The SYSnnnnn DD card is ignored if it exists. FORCE should not be used as the default.

BOTH

Language Environment creates a dynamic dump and, if a SYSnnnnn DD name exists, a dump is also written to the DD. BOTH should not be used as the default.

For U40xx ABENDS

The following suboptions are used for other U40xx ABENDS only. Existing SYSnnnnn DD statements are also honored.

TDUMP

Language Environment creates a dynamic dump automatically.

NOTDUMP

Language Environment does not create a dynamic dump.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- **Suggestions:**
 - Set up an *hlq* to which everyone can write.
 - Do not use FORCE or BOTH as the default for the U4039 ABENDS.
- The DYNDUMP runtime option is ignored under CICS.
- When an ABEND occurs during Language Environment initialization, the dynamic dump is not created if runtime options have not been processed yet.
- When the dynamic dump fails, messages are written to the operator's console or the job log (for batch). These are written by the IEATDUMP system service, by Language Environment, or by RACF®.
- When an ABEND has been issued without the DUMP option, no dump is generated.
- When Language Environment terminates with a U4038 abend, the U4038 abend is issued without the DUMP option. Therefore, no system dump is generated, and DYNDUMP does not collect a dump for this ABEND.
- The job name is taken from the JOBNAME system symbol.
 - A dump for a TSO application uses the user ID of the JOBNAME.
 - For a batch job, JOBNAME is taken from the JOB card in the JCL.
 - In the shell, JOBNAME is the user ID with a suffix.

Examples

- DYNDUMP=((smith,FORCE,NOTDUMP),OVR)

DYNDUMP

A dynamic dump is generated only for ABEND code U4039. Other SYSnnnnn DD cards are ignored. Other ABENDs might cause a dump to be created if a SYSnnnnn DD card exists. The dynamic dump data set name will be similar to SMITH.D012.T112245.JOB11.

- DYNDUMP=((smith,DYNAMIC,TDUMP),OVR)

A dynamic dump is created if no SYSnnnnn is specified and the ABEND code is U4039. The data set name will be similar to SMITH.D117.T235900.JOBZ2.

- DYNDUMP=((*TSOPREFIX,NODYNAMIC,TDUMP),OVR)

A dynamic dump is generated only for ABEND code U40xx. The data set name will be similar to SMITH.D287.T234560.JOBZ2.

- DYNDUMP=((*USERID,NODYNAMIC,TDUMP),OVR)

A dynamic dump for a U4039 ABEND is taken to SMITH.D109.T234512.JOBZ3.

- DYNDUMP=((*USERID.HOT.DUMPS,NODYNAMIC,TDUMP),OVR)
- DYNDUMP=((*TSOPRE.A1234567.B1234567,NODYNAMIC,TDUMP),OVR)

ENVAR

Derivation: ENvironmental VARiables

ENVAR sets the initial values for the environment variables that can later be accessed or changed using the C functions `getenv()`, `putenv`, `setenv`, and `clearenv`.

The set of environment variables established by the end of run-time option processing reflects all the various sources where environment variables are specified, rather than just the one source with the highest precedence. However, if a setting for the same environment variable is specified in more than one source, the setting with the highest precedence is used.

The `system()` function can be used to create a new environment. Environment variables in effect at the time of the POSIX `system()` call are copied to the new environment. The copied environment variables are treated the same as those found in the ENVAR run-time option on the command level.

When you specify the RPTOPTS run-time option, the output for the ENVAR runtime option contains a separate entry for each source where ENVAR was specified with the environment variables from that source.

Non-CICS default

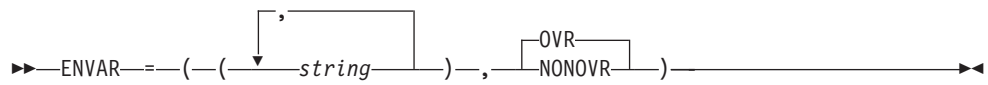
ENVAR=("",OVR)

CICS default

ENVAR=("",OVR)

AMODE 64 default

ENVAR=("",OVR)

Syntax**string**

Is of the form *name=value*, where *name* and *value* are sequences of characters that do not contain null bytes or equal signs. The string *name* is an environment variable, and *value* is its value.

Blanks are significant in both the *name=* and the *value* characters.

You can enclose the *string* in either single or double quotation marks to distinguish it from other strings. The *string* cannot contain DBCS characters. It can have a maximum of 250 characters.

You can specify multiple environment variables, separating the *name=value* pairs with commas. Quotation marks are required when specifying multiple variables.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

The environment variables apply to the enclave.

Usage notes

The ENVAR option functions independently of the POSIX runtime option setting.

C considerations

An application can access the environment variables using C function `getenv` or the POSIX variable *environ*, which is defined as:

```
extern char **environ;
```

Access through `getenv` is recommended, especially in a multithread environment.

HLLs can access the environment variables through standard C functions at enclave initialization and throughout the application's run. Access remains until the HLL returns from enclave termination. Environment variables that are propagated across the EXEC override those established by the ENVAR option. `getenv` serializes access to the environment variables.

C++ considerations

An application can access the environment variables using C function `getenv`.

HLLs can access the environment variables through standard C functions at enclave initialization and throughout the application's run.

For more information

For more information about the RPTOPTS runtime option, see “RPTOPTS” on page 107.

ERRCOUNT

Derivation: ERRor COUNTer

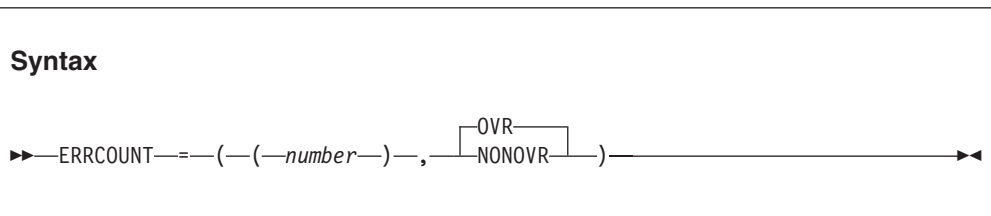
ERRCOUNT specifies how many conditions of severity 2, 3, and 4 can occur per thread before the enclave terminates abnormally. After the number specified in ERRCOUNT is reached, no further Language Environment condition management, including CEEHDLR management, is honored.

Non-CICS default

ERRCOUNT=((0),OVR)

CICS default

ERRCOUNT=((0),OVR)



number

The number of severity 2, 3, and 4 conditions per individual thread that can occur while this enclave is running. If the number of conditions exceeds *number*, the thread and enclave terminate abnormally.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

Synchronous signals that are associated with a condition of severity 2, 3, and 4 do not affect ERRCOUNT. Asynchronous signals do not affect ERRCOUNT.

Usage notes

- ERRCOUNT(0) means that the Language Environment condition handler will not terminate the task regardless of the severity 2, 3, or 4 conditions that are generated. This setting allows previously existing infinite loop or runaway task conditions to persist.
- ERRCOUNT only applies when conditions are handled by a user condition handler, signal catcher, PL/I on-units, or a language-specific condition handler. Language Environment does not count severity 0 or 1 messages. However, the COBOL specific runtime library does count its severity 1 (warning) messages. When the limit of 256 IGZnnnnW messages is reached, the COBOL library issues

message IGZ0041W, which indicates that the limit of warning messages was exceeded. Any further COBOL warning messages are suppressed.

C++ considerations

The ERRCOUNT option sets the threshold for the total number of severity 2, 3, and 4 synchronous conditions that can occur. Note that each thrown object is considered a severity 3 condition. However, this condition does not affect ERRCOUNT.

PL/I considerations

ERRCOUNT(0) is recommended for applications containing PL/I. Some conditions, such as ENDPAGE, can occur many times in an application. Use ERRCOUNT(0) to avoid unnecessary termination of your application.

PL/I MTF considerations

In a PL/I MTF application, ERRCOUNT sets the threshold for the total number severity 2, 3, and 4 synchronous conditions that can occur for each task. If the number of conditions exceeds the threshold, the application terminates normally.

For more information

- For a description of condition severities, see *z/OS Language Environment Programming Guide*.
- For more information about the CEEHDLR callable service, or the CEESGL callable service, see *z/OS Language Environment Programming Reference*.
- See *z/OS Language Environment Programming Guide* for more information about the facility ID part of messages.

ERRUNIT (Fortran only)

Derivation: ERRor UNIT

ERRUNIT identifies the unit number to which runtime error information is to be directed. This option is provided for compatibility with the VS Fortran version 2 runtime.

Non-CICS default

ERRUNIT=((6),OVR)

CICS default

ERRUNIT is ignored under CICS.

Syntax

```
▶▶ ERRUNIT = ( ( number ) , ( OVR | NONOVR ) ) ▶▶
```

number

A valid unit number in the range 0-99. The Language Environment message file and the file connected to the Fortran error message unit are the same.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

FILEHIST (Fortran only)

Derivation: FILE HISTory

FILEHIST specifies whether to allow the file definition of a file referred to by a ddname to be changed during runtime. This option is intended for use with applications called by Fortran that reallocate Fortran's supplied DD names.

Non-CICS default

FILEHIST=((ON),OVR)

CICS default

FILEHIST is ignored under CICS.

Syntax

```

  FILEHIST=((ON|OFF),OVR|NONOVR)
  
```

ON Causes the history of a file to be used in determining its existence. It checks to see whether:

- The file was ever internally opened (in which case it exists)
- The file was deleted by a CLOSE statement (in which case it does not exist).

OFF

Causes the history of a file to be disregarded in determining its existence.

If you specify FILEHIST(OFF), you should consider:

- **If you change file definitions during runtime**, the file is treated as if it were being opened for the first time. Before the file definition can be changed, the existing file must be closed.
- **If you do not change file definitions during runtime**, you must use STATUS='NEW' to re-open an empty file that has been closed with STATUS='KEEP', because the file does not appear to exist to Fortran.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

FILETAG (C/C++ only)

Derivation: FILE TAGging

FILETAG controls automatic text conversion and automatic file tagging for z/OS UNIX files. It activates the automatic file tagging, on the first write, of new or empty files open with fopen() or freopen(), or upon the first I/O to a pipe created with popen().

Recommendation: To use the runtime option properly, be familiar with the concept of file tagging, automatic conversion, and the CCSID. See *z/OS XL C/C++ Programming Guide* for more information.

Non-CICS default

FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR)

CICS default

FILETAG is ignored under CICS.

AMODE 64 default

FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR)

Syntax

```

▶▶ FILETAG = ( ( NOAUTOCVT | AUTOCVT ) , ( NOAUTOTAG | AUTOTAG ) ) , ( OVR | NONOVR )

```

NOAUTOCVT

Disables the following behavior:

AUTOCVT

Enables automatic text conversion for untagged files opened using `fopen()` or `freopen()`. The conversion for an untagged file will be from the program CCSID to the EBCDIC CCSID as specified by the `_BPXK_CCSDS` environment variable. If the environment variable is unset, a default CCSID pair is used. See *z/OS XL C/C++ Programming Guide* for more information about the `_BPXK_CCSDS` environment variable.

Restriction: Automatic conversion for untagged UNIX files can only take place between IBM-1047 and ISO8859-1 code sets. Other CCSID pairs are not supported. By default, automatic conversion for untagged files applies only to files opened in text mode. An untagged file opened in binary mode is not converted automatically. You can control this by using the `_EDC_AUTOCVT_BINARY` environment variable. For more information about the `_EDC_AUTOCVT_BINARY` environment variable, see *z/OS XL C/C++ Programming Guide*.

This suboption also indicates that the standard streams should be enabled for automatic text conversion to the EBCDIC IBM-1047 code page when they refer to an untagged terminal file (tty).

This suboption does not affect untagged files that are automatically tagged by the `AUTOTAG` suboption. A file that is automatically tagged is already enabled for automatic text conversion.

Requirement: The automatic text conversion is performed only if one of the following is also true:

- The `_BPXK_AUTOCVT` environment variable value is set to ON.
- The `_BPXK_AUTOCVT` environment variable is unset and `AUTOCVT(ON)` was specified in the active `BPXPRMxx` member on your system.

For more information about the `_BPXK_AUTOCVT` environment variable, see *z/OS XL C/C++ Programming Guide*.

NOAUTOTAG

Deactivates the automatic tagging of new or empty files.

FILETAG

AUTOTAG

Activates the automatic file tagging, on the first write, of new or empty files open with `fopen()`, `freopen()`, or `popen()`.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

FILETAG applies to the enclave. Nested enclaves do not inherit the setting of this runtime option. Files that are opened in the nested enclave are not affected.

Usage notes

- Avoid these situations:
 - Setting this runtime option at the system or region levels.
 - Setting this runtime option using `_CEE_RUNOPTS` in a default profile for the UNIX shell users.
 - Exporting `_CEE_RUNOPTS` that specifies this runtime option. It can cause unexpected behaviors for the unknowing user or application.
- The application programmer should define this runtime option with the assumption that the application is coded to behave based on the option's setting.
- The application programmer should specify this runtime option at compile time using `#pragma runopts` or at bind using a `CEEUOPT CSECT` that has been previously created.
- The application user should not override this runtime option because it can change the expected behavior of the application.
- The default CCSID pair is (1047,819), where 1047 indicates the EBCDIC IBM-1047 code page and 819 indicates the ASCII ISO8859-1 code page.
- Automatic text conversion is enabled for the standard streams only when the application has been `exec()`ed, for example, when the UNIX shell gives control to a program entered on the command line, and the standard stream file descriptors are already open, untagged and associated with a `tty`.
- For the UNIX shell-owned standard streams that are redirected at program execution time, the shell includes added environment variables that control whether the redirected streams are tagged. See *z/OS UNIX System Services Command Reference* for more information.
- Automatic tagging for a file is done at first write by the LFS. The CCSID used for the tag is the program CCSID of the current thread. Both text and binary files are tagged.
- When `FILETAG(,AUTOTAG)` is in effect, `fopen()` or `freopen()` of a file fails if it cannot determine whether the file exists or if it cannot determine the size.

HEAP

HEAP controls the allocation of the initial heap, controls allocation of additional heaps created with the `CEECRHP` callable service, and specifies how that storage is managed.

Heaps are storage areas where you allocate memory for user-controlled dynamically allocated variables such as:

- C variables allocated as a result of the `malloc()`, `calloc()`, and `realloc()` functions
- COBOL WORKING-STORAGE data items
- PL/I variables with the storage class CONTROLLED, or the storage class BASED

The HEAP option is always in effect. If you do not specify HEAP, Language Environment allocates the default value of heap storage when a call is made to get heap storage.

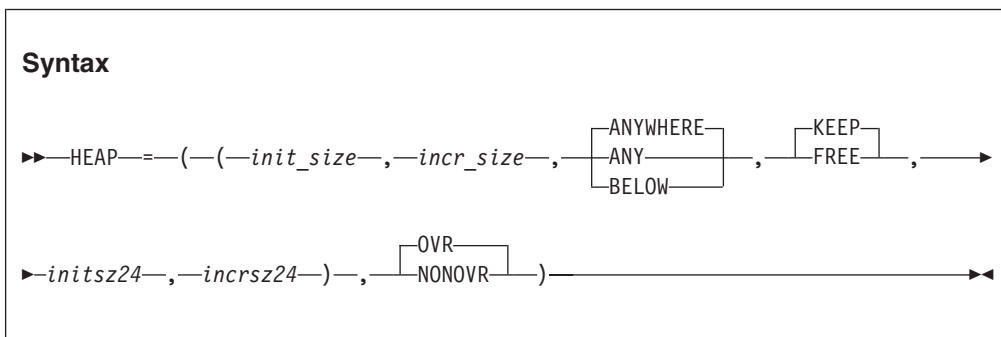
Language Environment does not allocate heap storage until the first call to get heap storage is made. You can get heap storage by using language constructs or by making a call to CEEGTST.

Non-CICS default

```
HEAP=((32K,32K,ANYWHERE,KEEP,8K,4K),OVR)
```

CICS default

```
HEAP=((4K,4080,ANYWHERE,KEEP,4K,4080),OVR)
```



init_size

Determines the minimum initial allocation of heap storage. Specify this value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr_size

Determines the minimum size of any subsequent increment to the heap storage. Specify this value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

ANYWHERE | **ANY**

Specifies that you can allocate heap storage anywhere in storage. If there is no available storage above the line, storage is acquired below the 16 MB line.

The only valid abbreviation of ANYWHERE is ANY.

BELOW

Specifies that you must allocate heap storage below the 16-MB line in storage that is accessible to 24-bit addressing.

KEEP

Specifies that storage allocated to HEAP increments is not released when the last of the storage is freed.

FREE

Specifies that storage allocated to HEAP increments is released when the last of the storage is freed.

HEAP

initsz24

Determines the minimum initial size of the heap storage that is obtained when an application running AMODE 24 (ALL31(OFF)) requests storage and ANYWHERE has been specified. An AMODE 31 application running with ALL31(OFF) uses the regular heap allocation size. Specify *initsz24* as *n*, *nK*, or *nM* number of bytes. The amount of storage is rounded up to the nearest multiple of 8 bytes.

initsz24 applies to the initial heap and other heaps created with the CEECRHP callable service that are not allocated strictly below the 16-MB line.

incrsz24

Determines the minimum size of any subsequent increment to the heap area that is obtained when an application running AMODE 24 (ALL31(OFF)) requests storage and ANYWHERE has been specified. An AMODE 31 application running with ALL31(OFF) uses the regular heap allocation size. Specify *incrsz24* as *n*, *nK*, or *nM* number of bytes. The amount of storage is rounded up to the nearest multiple of 8 bytes.

The *incrsz24* applies to the initial heap and other heaps created with the CEECRHP callable service that are not allocated strictly below the 16-MB line.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

Both the initial HEAP allocation and HEAP increments are rounded to the next higher multiple of 8 bytes (not 4K bytes). If HEAP(0) is specified the initial HEAP is obtained on the first use and will be based on the increment size.

If HEAP(„ANYWHERE,„) is in effect, the maximum size of a heap segment is 1 gigabyte (1024 MB). These restrictions are subject to change from one release of CICS to another.

The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16 bytes CICS storage check zone. Without this accommodation, an extra page of storage is allocated (only when the storage allocation is below the 16MB line).

z/OS UNIX considerations

The HEAP option applies to the enclave.

Under z/OS UNIX, heap storage is managed at the thread level using `pthread_key_create`, `pthread_setspecific`, and `pthread_getspecific`.

Usage notes

- Applications running in AMODE 24 that request heap storage get the storage below the 16 MB line regardless of the setting of ANYWHERE | BELOW.
- For PL/I, the only case in which storage is allocated above the line is when all of the following conditions exist:
 - The user routine requesting the storage is running in 31-bit addressing mode.
 - HEAP(„ANY,„) is in effect.
 - The main routine is AMODE 31.

AMODE(31) and RMODE(31) are the default settings for Enterprise PL/I applications. To run an application in AMODE(24) you must:

1. Compile all PL/I source with the compiler option NORENT
2. Link with the SIBMAM24 data set concatenated in front of the SCEELKED data set
3. Run with the Language Environment runtime option ALL31(OFF),HEAP(„BELOW,,,), and STACK(„BELOW,,,)

COBOL considerations

You can use the HEAP option to provide function similar to the VS COBOL II space management tuning table.

PL/I MTF considerations

In a PL/I MTF application, HEAP specifies the heap storage allocation and management for a PL/I main task.

Performance considerations

To improve performance, use the storage report numbers generated by the RPTSTG runtime option as an aid in setting the initial and increment size for HEAP.

For more information

- See *z/OS Language Environment Programming Guide* for more information about Language Environment heap storage or about specifying runtime options at application invocation.
- For more information about the CEECRHP callable service, or the CEEGTST callable service, see *z/OS Language Environment Programming Reference*.
- See “RPTSTG” on page 108 for more information about the RPTSTG runtime option.

HEAP64 (AMODE 64 only)

HEAP64 controls the allocation of user heap storage for AMODE 64 applications and specifies how that storage is managed.

Heaps are storage areas that contain user-controlled dynamically allocated variables or data. An example is C data allocated as a result of the `malloc()`, `calloc()`, and `realloc()` functions.

AMODE 64 default

```
HEAP64=((1M,1M,KEEP,32K,32K,KEEP,4K,4K,FREE),OVR)
```

Syntax

```

▶▶—HEAP64—==—(—(—init64—,—incr64—,—
    [KEEP]
    [FREE]
    [FILL]
    —,—init31—,—incr31—▶▶
▶,—
    [KEEP]
    [FREE]
    —,—init24—,—incr24—,—
    [FREE]
    [KEEP]
    )—,—
    [OVR]
    [NONOVR]
    )—▶▶

```

init64

Determines the initial allocation of heap storage obtained above the 2G bar. Specify this value as *nM* bytes of storage. If a value of 0 or less is specified, the default is used.

incr64

Determines the minimum size of any subsequent increment to the user heap storage obtained above the 2G bar. Specify this value as *nM* bytes of storage. If a value less than 0 is specified, the default is used.

KEEP

Specifies that an increment to user heap storage is not released when the last of the storage within that increment is freed.

FREE

Specifies that an increment to user heap storage is released when the last of the storage within that increment is freed.

FILL

Specifies that an increment to user heap storage is released when the last of the storage within that increment is freed. In addition, when a storage request results in a new increment segment being created which is greater than the **incr64** size, the entire segment is filled by the single storage request. This option is available only for user heap storage above the bar.

init31

Determines the minimum initial size of user heap storage that is obtained above the 16MB line and below the 2GB bar. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value of 0 or less is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr31

Determines the minimum size of any subsequent increment to user heap storage that is obtained above the 16MB line and below the 2GB bar. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value less than 0 is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

init24

Determines the minimum initial size of user heap storage that is obtained below the 16MB line. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value of 0 or less is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr24

Determines the minimum size of any subsequent increment to user heap storage that is obtained below the 16MB line. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value less than 0 is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

- In a multithreaded environment, user heap storage is shared by all threads the process.

- Heap storage is managed at the thread level using `pthread_key_create`, `pthread_setspecific`, and `pthread_getspecific`.

Performance considerations

You can improve performance with the HEAP64 runtime option by specifying values that minimize the number of times the operating system allocates storage. See “RPTSTG” on page 108 for information about how to generate a report you can use to determine the optimum values for the HEAP64 runtime option.

For more information

- For more information about Language Environment heap storage, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.
- For more information about the RPTSTG runtime option, see “RPTSTG” on page 108.

HEAPCHK

Derivation: HEAP storage CHeCking

Use HEAPCHK to run additional heap check tests.

Non-CICS default

```
HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR)
```

CICS default

```
HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR)
```

Amode 64 default

```
HEAPCHK=((OFF,1,0,0,0,1024,0,1024,0),OVR)
```

Syntax

```

▶▶HEAPCHK=-(OFF
  ( ON , frequency , delay , call depth ,
  pool call depth , number of entries , pool number ,
  number of entries 31 , pool number 31 ) , OVR
  ( NONOVR ) )

```

OFF

Indicates that no heap checking or tracing is done regardless of the values of the remaining suboptions.

ON Indicates that heap checking or tracing is activated based on the values of the remaining suboptions.

frequency

The frequency at which the user heap is checked for damage to the heap control information. It is specified as *n*, *nK* or *nM*. A value of one (1) is the default and causes the heap to be checked at each call to a Language Environment heap storage management service. A value of *n* causes the user

HEAPCHK

heap to be checked at every *n*th call to a Language Environment heap storage management service. A value of zero results in no check for damage to the user heap.

delay

A value that causes a delay before user heap is checked for damage, and is specified in *n*, *n*K or *n*M. A value of zero (0) is the default and causes the heap checking to begin with the first call to a Language Environment heap storage management service. A value of *n* causes the heap checking to begin following the *n*th call to a Language Environment heap storage management service.

call depth

Specifies the depth of calls displayed in the traceback for the heap storage diagnostics report. A value of zero is the default that turns heap storage diagnostics reporting off.

The heap storage diagnostics report consists of a set of tracebacks. Each traceback is a snapshot of the stack (to a specified call depth) for each request to obtain user heap storage that has not had a corresponding free request.

pool call depth

Specifies the depth of calls in the traceback for each trace entry of a heap pools trace. A value of zero is the default that turns heap pools tracing off.

The heap pools trace is an in-core wrapping trace. Each heap pool has a separate trace table. The heap pools trace is only formatted from a system dump using the IPCS Verbexit LEDATA.

number of entries

Specifies the number of entries to be recorded in one heap pool trace table for the main user heap in the application. Each pool has its own trace table. If the number of entries is 0, the heap pool trace table is not generated.

pool number

Specifies which pools are traced for the main user heap in the application. You can either trace one pool or all pools. The value should be a valid pool number from 1 to 12. If the pool number is 0, all pools will be traced.

number of entries 31

Specifies the number of entries to be recorded in one heap pool trace table when an application is using heap storage from 31-bit addressable storage (`__malloc31()`). Each pool has its own trace table. If the number of entries is 0, the heap pool trace table is not generated. This value is only supported in an AMODE64 environment.

pool number 31

Specifies which pools are traced when an application is using heap storage from 31-bit addressable storage (`__malloc31()`). You can trace either one pool or all pools. The value should be a valid pool number from 1 to 12. If the pool number is 0, all pools will be traced. This value is only supported in an AMODE64 environment.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- If HEAPCHK(ON) is used with STORAGE(heap_free_value), the free areas of the heap will also be checked.

- If HEAPCHK(ON) is specified, this will result in a performance degradation due to the additional error checking that is performed.
- A U4042 abend dump will be generated when an error is detected, but no CEEDUMP will be produced.
- To request only a heap storage diagnostics report, you must specify a zero for frequency, a zero for pool call depth and a number n greater than zero for call depth. For example, you could specify HEAPCHK(ON,0,0,10,0,1024,0,1024,0).
Recommendation: Use a value of 10 to ensure an adequate call depth is displayed so that you can identify the storage leak.
- To request heap pools tracing, set *pool call depth* to a nonzero value and *number of entries* (for AMODE 64 applications, *number of entries*, *number of entries 31*, or both) to a nonzero value. To request only heap pools tracing, in addition, set *frequency* to zero and *call depth* to zero. The heap pools trace is only formatted from a system dump using the IPCS Verbexit LEDATA.
Recommendation: Use a value of 10 to ensure an adequate call depth is displayed.
- For AMODE 64 applications, number of entries and pool number control tracing for the set of heap pools located in storage above the 2GB bar. Number of entries 31 and pool number 31 control tracing for the set of heap pools located in storage above the 16MB line and below the 2GB bar. Pool call depth applies to both sets of heap pools.
- Under normal termination conditions, when the call depth is greater than zero, the heap storage diagnostics report is written to the CEEDUMP report. This is independent of the TERMTHDACT setting.
- If a heap storage diagnostics report is desired while calling CEE3DMP, you must specify the BLOCKS option.

For more information

See *z/OS Language Environment Debugging Guide* for more information about creating and using the heap storage diagnostics report.

HEAPPOOLS (C/C++ and Enterprise PL/I only)

Derivation: HEAP storage POOLS

The HEAPPOOLS runtime option is used to control an optional heap storage management algorithm known as heap pools. This algorithm is designed to improve performance of multithreaded C/C++ applications with high usage of `malloc()`, `_malloc31()`, `calloc()`, `realloc()`, `free()`, `new()`, and `delete()`. When active, heap pools can eliminate contention for heap storage.

Non-CICS default

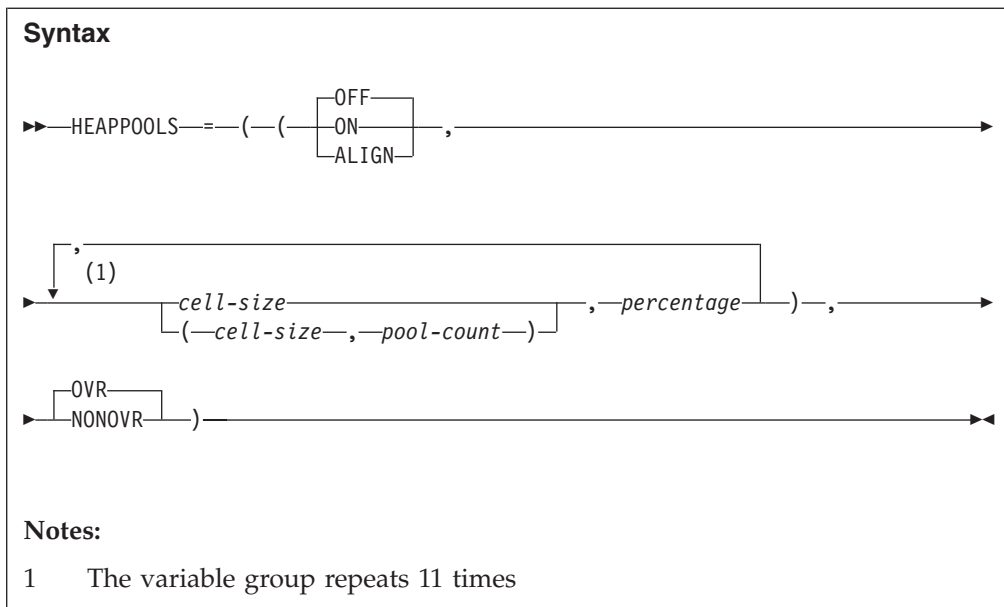
```
HEAPPOOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048,
10,0,10,0,10,0,10,0,10,0,10,0,10),OVR)
```

CICS default

```
HEAPPOOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048,
10,0,10,0,10,0,10,0,10,0,10,0,10),OVR)
```

AMODE 64 default

```
HEAPPOOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048,
10,0,10,0,10,0,10,0,10,0,10,0,10),OVR)
```



OFF

Specifies that Language Environment does not use the Heap Pools Manager.

ON

Specifies that Language Environment does use the Heap Pool Manager to manage heap storage requests against the initial heap.

ALIGN

Specifies that Language Environment structures the storage for cells in a heap pool so that a cell less than or equal to 248 bytes does not cross a cache line. For cells larger than 248 bytes, two cells never share a cache line.

cell-size

The size of cells in a heap pool. The cell size must be a multiple of 8 within a range from 8 to 65536. Cell sizes 1K, to 64K are also allowed.

pool-count

The number of pools to be created for the cell size. The pool-count must be in a range from 1 to 255.

percentage

Percentage of the HEAP runtime option init size value to be used as the size for the heap pool and any extents. The percentage must be in a range from 1 to 90.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- To use less than the supported number of heap pools, specify 0 for the cell size after the last heap pool to be used. For example if four heap pools are desired, use 0 for the fifth cell size when setting the HEAPPOOLS runtime option.
- If the percentage of the HEAP runtime option init size values does not allow for at least one cell, the system automatically adjusts the percentage to enable four cells to be allocated.

- The sum of the percentages may be more or less than 100 percent. This can cause the allocation of a heap pool to require the allocation of a heap increment to satisfy the request.
- Each heap pool is allocated on an as-needed basis. The allocation of a heap pool can result in the allocation of a heap increment to satisfy the request.
- For tuning and performance information, see *z/OS Language Environment Programming Guide*.
- Heap pools and extents are not released back to the heap, and cell sizes are fixed, so care should be taken when specifying the HEAPPOOLS runtime option to avoid wasting storage.
- The HEAPPOOLS runtime option has no effect when the initial heap is allocated below the 16-MB line. This would be the case when BELOW is specified as the location on the HEAP runtime option.
- The FREE suboption on the HEAP runtime option has no effect on the initial heap or any extents in which a heap pool resides. Each cell in a heap pool can be freed, but the heap pool itself is only released back to the system at enclave termination.
- Mixing of the storage management AWIs (CEEGTST(), CEEFRST() and CEECZST()) and the C/C++ intrinsic functions (malloc(), calloc(), realloc() and free()) is not supported when operating on the same storage address. For example, if you request storage using CEEGTST(), then you may not use free() to release the storage.
- The HEAPPOOLS runtime option applies to the enclave.
- Using the ALIGN suboption might cause an increase in the amount of heap storage used by an application.
- Examine the storage report and adjust storage tuning when first using the ALIGN suboption.
- The RPTSTG runtime option will indicate HEAPPOOLS as one of the runtime options which can be adjusted.
- The HEAPCHK runtime option will indicate that individual cells in the cell pools controlled by the HEAPPOOLS runtime option are not validated. It is the heap pool itself which is validated, as it is the actual storage managed by the regular storage manager.
- If you specify the RPTSTG runtime option while using HEAPPOOLS, extra storage is obtained from the ANYHEAP and is used to complete the storage report on heappools. This extra storage is only allocated when both HEAPPOOLS and RPTSTG are used.
- When *cell-size* is specified without parenthesis, the default value of *pool-count* is 1 rather than being picked up from an earlier setting of *pool-count*. For example, specifying 128 is treated like (128,1).
- When *cell-size* is specified within parenthesis, *pool-count* must also be specified.
- When *pool-count* is greater than 1, the size of each heap pool extent is determined by dividing the heap allocation for the *cell-size* by the *pool-count*.
- HEAPPOOLS runtime option can be used by AMODE 64 applications to manage user heap storage above the 16MB line and below the 2GB bar.

Performance considerations

- To improve the effectiveness of the heap pools algorithm, use the storage report numbers generated by the RPTSTG runtime option as an aid in determining optimum cell sizes, percentages, and the initial heap size.

HEAPPOOLS

- Use caution when using cells larger than 2K. Large gaps between cell sizes can lead to a considerable amount of storage waste. Properly tuning cell sizes with the help of RPTSTG is necessary to control the amount of virtual storage needed by the application.
- When there are many successful get requests for the same size cell and the maximum elements used in the cell pool is high, this can be an indication that there is excessive contention allocating elements in the cell pool. Specifying *pool-count* greater than 1 might help relieve some of this contention. Multiple pools are allocated with the same cell size and a portion of the threads are assigned to allocate cells out of each of the pools.

Examples

Specifying HEAPPOOLS(ON,(8,4),20,(16,2),10) results in:

- Four cell pools being allocated for 8-byte cells with each pool using 5 percent of the heap allocation
- Two cell pools being allocated for 16-byte cells with each pool using 5 percent of the heap allocation.

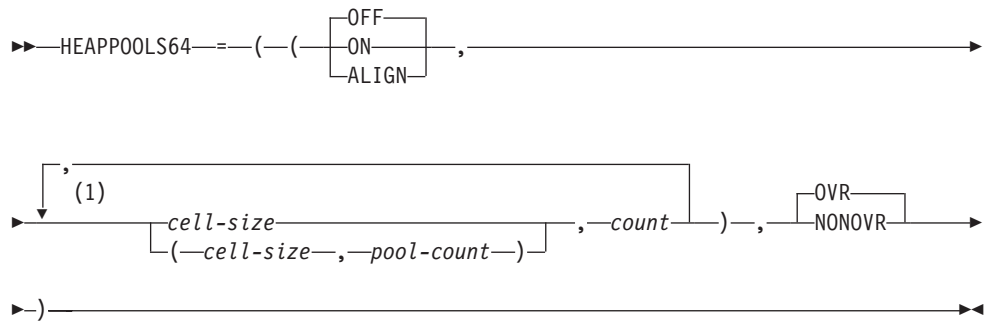
HEAPPOOLS64 (AMODE 64 only)

Derivation: HEAP storage POOLS for AMODE 64

The HEAPPOOLS64 runtime option is used to control an optional user heap storage management algorithm, known as heap pools. This algorithm is designed to improve the performance of multithreaded C/C++ applications with a high frequency of calls to malloc(), calloc(), realloc(), free(), and operators new and delete. When active, heap pools virtually eliminates contention for user heap storage.

AMODE 64 default

```
HEAPPOOLS64=((OFF,8,4000,32,2000,128,700,256,350,  
1024,100,2048,50,3072,50,4096,50,8192,25,16384,10,32768,5,65536,5),OVR)
```

Syntax**Notes:**

1 repeats 11 times

OFF

Specifies that the AMODE 64 heappools algorithm is not used.

ON Specifies that the AMODE 64 heappools algorithm is used.

ALIGN

Specifies that Language Environment structures the storage for cells in a heap pool so that a cell less than or equal to 240 bytes does not cross a cache line. For cells larger than 240 bytes, two cells never share a cache line.

cell-size

Specifies the size of the cells in a heap pool, specified as n or nK. The cell size must be a multiple of 8, with a maximum of 65536 (64K).

pool-count

The number of pools to create for the cell size. The pool-count must be in a range from 1 to 255.

count

Specifies the number of cells of the corresponding size to be allocated initially. The minimum cell count is 4.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- **Restriction:** HEAPPOOLS64 only manages storage above the 2G bar.
- Cell pool sizes should be specified in ascending order.
- To use less than twelve heap pools, specify 0 for the cell size after the last heap pool to be used. For example if four heap pools are desired, use 0 for the fifth cell size when setting the HEAPPOOLS64 runtime option.
- Each heap pool is allocated as needed. The allocation of a heap pool can result in the allocation of a heap increment to satisfy the request.
- Using the ALIGN suboption might cause an increase in the amount of heap storage used by an application.

HEAPPOOLS64

- Examine the storage report and adjust storage tuning when first using the ALIGN suboption.
- The HEAPCHK runtime option does not validate individual heap pool cells.
- If you specify the RPTSTG runtime option while using HEAPPOOLS64, extra storage is obtained from the LIBHEAP64 and is used to complete the storage report on heappools. This extra storage is only allocated when both HEAPPOOLS64 and RPTSTG are used.
- When *cell-size* is specified without parenthesis, the default value of *pool-count* is 1 rather than being picked up from an earlier setting of *pool-count*. For example, specifying 128 is treated like (128,1).
- When *cell-size* is specified within parenthesis, *pool-count* must also be specified.
- When *pool-count* is greater than 1, the size of each heap pool extent is determined by dividing the heap allocation for the cell-size by the *pool-count*.
- HEAPPOOLS runtime option can be used by AMODE 64 applications to manage user heap storage above the 16MB line and below the 2GB bar.

Performance considerations

- To improve the effectiveness of the heap pools algorithm, use the storage report numbers generated by the RPTSTG runtime option as an aid in determining optimum cell sizes, cell count, and the initial heap size.
- Use caution when using cells larger than 2K. Large gaps between cell sizes can lead to a considerable amount of storage waste. Properly tuning cell sizes with the help of RPTSTG is necessary to control the amount of virtual storage needed by the application.
- When there are many successful get requests for the same size cell and the maximum elements used in the cell pool is high, this can be an indication that there is excessive contention allocating elements in the cell pool. Specifying *pool-count* greater than 1 might help relieve some of this contention. Multiple pools are allocated with the same cell size and a portion of the threads are assigned to allocate cells out of each of the pools.

Examples

Specifying HEAPPOOLS64(ON,(8,4),1000,(16,2),500) results in:

- Four cell pools being allocated for 8-byte cells with each pool extent containing 250 cells.
- Two cell pools being allocated for 16-byte cells with each pool containing 250 cells.

INFMSGFILTER

Derivation

INFormational MeSsaGe FILTER

During normal operations, there are times when long lists of informational messages are written to the Language Environment MSGFILE. These messages are not limited to Language Environment (CEE) messages. Informational messages may also be written, using the CEEMSG interface, by other IBM program products or user-written applications. If these messages are routed to the user's terminal, then the user must constantly clear them. If the messages are saved to a data set, they take up disk space and may interfere with a user browsing the output looking for a specific message. INFMSGFILTER allows the user to activate a filter that

eliminates the unwanted and unnecessary informational messages. All informational messages, whether generated by Language Environment or any other source, are suppressed when the INFOMSGFILTER=(ON) option is in effect.

Non-CICS default

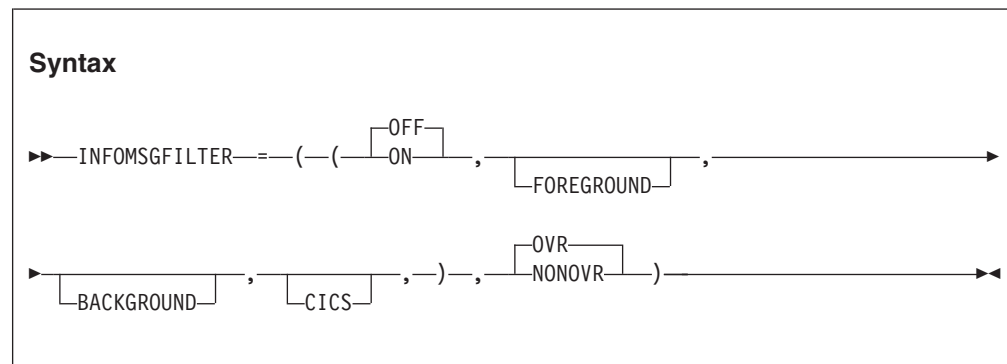
INFOMSGFILTER=((OFF,,,,),OVR)

CICS default

INFOMSGFILTER=((OFF,,,,),OVR)

Amode 64 default

INFOMSGFILTER=((OFF,,,,),OVR)



OFF

Turns off the filtering of messages for all environments.

ON

Turns on the filtering of messages for the specified environments.

FOREGROUND

Selecting this keyword turns message filtering on for the following environments:

- TSO
- CMS
- z/OS UNIX

BACKGROUND

Selecting this keyword turns message filtering on for the following environments:

- MVS Batch
- CMS Batch

CICS

Selecting this keyword turns message filtering on in the CICS environment. This is ignored for AMODE 64 applications.

Note: These three keywords are not positional; you can specify them in any order. The fourth comma is required when coding this option at the region level, even though there is no keyword to fill its position. This position is reserved by IBM for future use.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

INQPCOPN (Fortran only)

Derivation: INquire the Pre-Conected units that are OPeNed

INQPCOPN controls whether the OPENED specifier on an INQUIRE by unit statement can be used to determine whether a preconnected unit has had any I/O statements directed to it.

Non-CICS default

INQPCOPN=((ON),OVR)

CICS default

INQPCOPN is ignored under CICS.

Syntax

```

  INQPCOPN = ( ( ON | OFF ) , ( OVR | NONOVR ) )
  
```

ON Causes the running of an INQUIRE by unit statement to provide the value *true* in the variable or array element given in the OPENED specifier if the unit is connected to a file. This includes the case of a preconnected unit, which can be used in an I/O statement without running an OPEN statement, even if no I/O statements have been run for that unit.

OFF

Causes the running of an INQUIRE by unit statement to provide the value *false* for the case of a preconnected unit for which no I/O statements other than INQUIRE have been run.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

INTERRUPT

INTERRUPT causes attention interrupts recognized by the host system to be recognized by Language Environment after the Language Environment environment has been initialized. The way you request that an attention interrupt varies from operating system to operating system. When you request the interrupt, you can give control to your application or to a debug tool.

Non-CICS default

INTERRUPT=((OFF),OVR)

CICS default

INTERRUPT is ignored under CICS.

Syntax

```
▶▶—INTERRUPT—==—(—(—OFF—)  
ON—),—OVR—  
NONOVR—)▶▶
```

OFF

Specifies that Language Environment does not recognize attention interrupts.

ON

Specifies that Language Environment recognizes attention interrupts. In addition, if you specified the TEST(ERROR) or TEST(ALL) run-time option, the interrupt causes the debug tool to gain control.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

The INTERRUPT option applies to the enclave. However, only one thread in the enclave is affected for a particular attention interrupt.

Usage notes**PL/I considerations**

Language Environment supports the PL/I method of polling code. The PL/I routine must be compiled with the INTERRUPT compiler option in order for the INTERRUPT runtime option to have an effect.

PL/I MTF considerations

To receive the attention interrupt, the PL/I program must be compiled with the INTERRUPT compiler option, and the INTERRUPT runtime option must be in effect.

The INTERRUPT option applies to the enclave. However, only one thread in the enclave is affected for a particular attention interrupt.

For more information

- For more information about the TEST runtime option, see “TEST | NOTEST” on page 127.
- For more information about the POSIX runtime option, see “POSIX” on page 103.

IOHEAP64 (AMODE 64 only)

Derivation: IO HEAP storage for AMODE 64

IOHEAP64 controls the allocation of I/O heap storage for AMODE 64 applications and specifies how that storage is managed. Language Environment uses this storage when performing I/O for AMODE 64 applications.

AMODE 64 default

```
IOHEAP64=((1M,1M,FREE,12K,8K,FREE,4K,4K,FREE),OVR)
```

Syntax

```

▶▶ IOHEAP64 = ( ( ( init64 , incr64 , KEEP / FREE , init31 , incr31
▶ , KEEP / FREE , init24 , incr24 , FREE / KEEP ) , OVR / NONOVR )

```

init64

Determines the initial allocation of I/O heap storage obtained above the 2G bar. Specify this value as *nM* bytes of storage. If a value of 0 or less is specified, the default is used.

incr64

Determines the minimum size of any subsequent increment to the I/O heap storage obtained above the 2G bar. Specify this value as *nM* bytes of storage. If a value less than 0 is specified, the default is used.

KEEP

Specifies that an increment to I/O heap storage is not released when the last of the storage within that increment is freed.

FREE

Specifies that an increment to I/O heap storage is released when the last of the storage within that increment is freed.

init31

Determines the minimum initial size of I/O heap storage that is obtained above the 16MB line and below the 2GB bar. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value of 0 or less is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr31

Determines the minimum size of any subsequent increment to I/O heap storage that is obtained above the 16MB line and below the 2GB bar. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value less than 0 is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

init24

Determines the minimum initial size of I/O heap storage that is obtained below the 16MB line. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value of 0 or less is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr24

Determines the minimum size of any subsequent increment to I/O heap storage that is obtained below the 16MB line. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value less than 0 is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Performance considerations

To improve performance, use the storage report numbers generated by the RPTSTG runtime option as an aid in setting the initial and increment sizes for IOHEAP64.

For more information

See “RPTSTG” on page 108 for more information about the RPTSTG runtime option.

LIBHEAP64 (AMODE 64 only)

Derivation: LIBRARY HEAP storage for AMODE 64

The LIBHEAP64 runtime option controls the allocation of heap storage used by Language Environment for AMODE 64 applications and specifies how that storage is managed.

AMODE 64 default

LIBHEAP64=((1M,1M,FREE,16K,8K,FREE,8K,4K,FREE),OVR)

Syntax

```

▶▶ LIBHEAP64==((--(---init64---,---incr64---,---KEEP---
   |---FREE---,---init31---,---
▶---incr31---,---KEEP---
   |---FREE---,---init24---,---incr24---,---FREE---
   |---KEEP---)---,---OVR---
   |---NONOVR---
▶)---▶▶

```

init64

Determines the initial allocation of library heap storage obtained above the 2G bar. Specify this value as *nM* bytes of storage. If a value of 0 or less is specified, the default is used.

incr64

Determines the minimum size of any subsequent increment to the library heap storage obtained above the 2G bar. Specify this value as *nM* bytes of storage. If a value less than 0 is specified, the default is used.

KEEP

Specifies that an increment to library heap storage is not released when the last of the storage within that increment is freed.

FREE

Specifies that an increment to library heap storage is released when the last of the storage within that increment is freed.

init31

Determines the minimum initial size of library heap storage that is obtained above the 16MB line and below the 2GB bar. This value can be specified as *n*,

LIBHEAP64

nK, or *nM* number of bytes. If a value of 0 or less is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr31

Determines the minimum size of any subsequent increment to library heap storage that is obtained above the 16MB line and below the 2GB bar. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value less than 0 is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

init24

Determines the minimum initial size of library heap storage that is obtained below the 16MB line. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value of 0 or less is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incr24

Determines the minimum size of any subsequent increment to library heap storage that is obtained below the 16MB line. This value can be specified as *n*, *nK*, or *nM* number of bytes. If a value less than 0 is specified, the default is used. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Performance considerations

To improve performance, use the storage report numbers generated by the RPTSTG runtime option as an aid in setting the initial and increment sizes for LIBHEAP64.

For more information

- For more information about heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

LIBSTACK

Derivation: LIBrary STACK storage

LIBSTACK controls the allocation of the thread's library stack storage. This stack is used by Language Environment and HLL library routines that require save areas below the 16MB line.

Non-CICS default

LIBSTACK=((4K,4K,FREE),OVR)

CICS default

LIBSTACK=((32,4080,FREE),OVR)

Syntax

```

▶▶ LIBSTACK=-( -( -init_size-, -incr_size-, [FREE-KEEP] )-,
[OVR-NONOVR] )

```

init_size

Determines the minimum size of the initial library stack segment. The storage is contiguous.

Specify *init_size* as *n*, *nK*, or *nM* bytes of storage. *init_size* can be preceded by a minus sign. In environments other than CICS, if you specify a negative number, all available storage minus the amount specified is used for the initial stack segment.

In non-CICS environments, an *init_size* of 0 or -0 requests half of the largest block of contiguous storage below the 16-MB line. In addition when STACK(„ANY,,”) is in effect, Language Environment does not acquire the initial library stack segment until the first program that requires LIBSTACK runs.

Language Environment allocates the storage rounded up to the nearest multiple of 8 bytes.

incr_size

Determines the minimum size of any subsequent increment to the library stack area. Specify this value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is the larger of 2 value (the *incr_size* or the requested size) rounded up to the nearest multiple of 8 bytes.

If you do not specify *incr_size*, Language Environment uses the Non-CICS default setting of 4K. If *incr_size*=0, Language Environment gets only the amount of storage needed at the time of the request, rounded up to the nearest multiple of 8 bytes.

The requested size is the amount of storage a routine needs for a stack frame. For example, if the requested size is 9000 bytes, *incr_size* is specified as 8K, and the initial stack segment is full, then Language Environment gets a 9000 byte stack increment from the operating system to satisfy the request. If the requested size is smaller than 8K, Language Environment gets an 8K stack increment from the operating system.

FREE

Specifies that Language Environment releases storage allocated to LIBSTACK increments when the last of the storage in the library stack is freed. The initial library stack segment is not released until the enclave terminates.

KEEP

Specifies that Language Environment does not release storage allocated to LIBSTACK increments when the last of the storage is freed.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

LIBSTACK

CICS considerations

- If ALL31(ON) is specified, LIBSTACK is allocated above the 16MB line.
- The initial and increment sizes for LIBSTACK are rounded to the next higher multiple of 8 bytes.
- The minimum initial size is 32 bytes; the minimum increment size is 4080.
- The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16-bytes CICS storage check zone. Without this accommodation, an extra page of storage is allocated (only when the storage allocation is below the 16-MB line).

z/OS UNIX considerations

The LIBSTACK option sets the library stack characteristics on each thread.

The recommended setting for LIBSTACK under z/OS UNIX is
LIBSTACK=((4K,4K,FREE),OVR).

Performance considerations

To improve performance, use the storage report numbers generated by the RPTSTG run-time option as an aid in setting the initial and increment size for LIBSTACK.

For more information

- For more information about the RPTSTG runtime option “RPTSTG” on page 108.
- For more information about using the storage reports generated by the RPTSTG runtime option to tune the stacks, see *z/OS Language Environment Programming Guide*.

MSGFILE

Derivation: MeSsaGe FILE

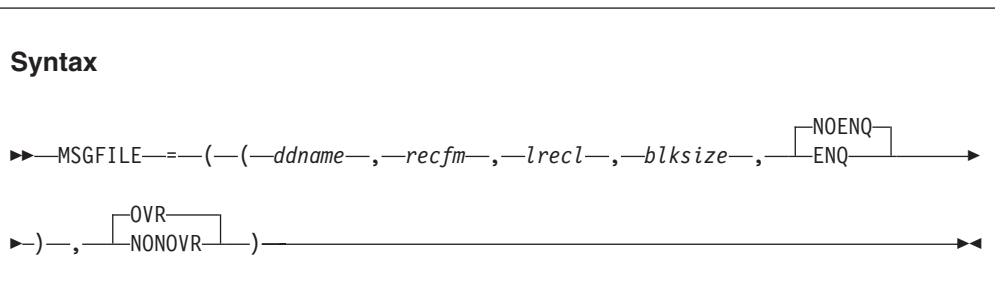
MSGFILE specifies the *ddname* of the file where all runtime diagnostics and reports generated by the RPTOPTS and RPTSTG run-time options are written. MSGFILE also specifies the *ddname* for CEEMSG and CEEMOUT callable services.

Non-CICS default

MSGFILE=((SYSOUT,FBA,121,0,NOENQ),OVR)

CICS default

MSGFILE is ignored under CICS.



ddname

The *ddname* of the runtime diagnostics file.

recfm

The default record format (RECFM) value for the message file. *recfm* is used when this information is not available either in a file definition or in the label of an existing file. The following values are acceptable: F, FA, FB, FBA, FBS, FBSA, U, UA, V, VA, VB, and VBA.

lrecl

The default record length (LRECL) value for the message file. *lrecl* is used when this information is not available either in a file definition or in the label of an existing file. *lrecl* is expressed as bytes of storage.

The *lrecl* value (whether from MSGFILE or from another source) cannot exceed the *blksize* value, whose maximum value is 32760. For variable-length record formats, the *lrecl* value is limited to the *blksize* value minus 4.

blksize

The default block size (BLKSIZE) value for the message file. *blksize* is used when this information is not available either in a file definition or in the label of an existing file. *blksize* is expressed as bytes of storage.

blksize (whether from MSGFILE or from another source) cannot exceed 32760.

NOENQ

Serialization around writes to the message file destination specified *ddname* are not performed.

ENQ

Specifies that serialization is performed around writes to the *ddname* specified.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

This runtime output under CICS is directed to a transient data queue named CESE.

z/OS UNIX considerations

The MSGFILE option specifies the *ddname* of the diagnostic file for the enclave. When multiple threads write to the message file, the output is interwoven by line. To group lines of output, serialize MSGFILE access (by using a mutex, for example).

When z/OS UNIX is available and the MSGFILE option specifies a *ddname* nominating a POSIX file, Language Environment uses POSIX services to write the message file. A *ddname* nominates a POSIX file using the keyword *PATH=*.

z/OS UNIX must be available on the underlying operating system for the MSGFILE option to write to a POSIX file. If the *ddname* nominates a POSIX file and z/OS UNIX is not present, then Language Environment tries to dynamically allocate an MVS file to be used as the message file.

If the message file is allocated (whether POSIX or z/OS), Language Environment directs the output to this file. If the current message file is not allocated, and the

application carries out a `fork()/exec`, `spawn()`, or `spawnp()`, Language Environment checks whether File Descriptor 2 exists. If it does exist, then Language Environment uses it; otherwise, Language Environment dynamically allocates the message file to the POSIX file system and attempts to open the file `SYSDOUT` in the current working directory; or, if there is no current directory, then in the directory `/tmp`.

Usage notes

- The ENQ suboption should only be used where multiple Language Environment environments are running in the same address space and are sharing the same MSGFILE destination. An example would be a batch job which uses ATTACH to create some number of subtasks. Each of these tasks is potentially a distinct Language Environment environment all running with the same default MSGFILE parameters. In this example, each of these environments will share the same MSGFILE destination. To avoid conflicts while writing to the shared MSGFILE destination, it is recommended that the ENQ suboption be used for each MSGFILE destination that will be shared. Using different *ddname* for each environment would remove the need to use the ENQ suboption.
- HLL compiler options, such as the COBOL OUTDD compiler option, can affect whether your runtime output goes to MSGFILE *ddname*.
- Use commas to separate suboptions of the MSGFILE runtime option. If you do not specify a suboption but do specify a subsequent one, you must still code the comma to indicate its omission. However, trailing commas are not required. If you do not specify any suboptions, either of the following is valid: MSGFILE or MSGFILE().
- If there is no block size in the MSGFILE runtime option, in a file definition, or in the label of an existing file, block size is determined as follows:
 - For a *recfm* value that specifies unblocked fixed-length format records (F or FA) or undefined-format records (U or UA), the *blksize* value is the same as the *lrecl* value.
 - For a *recfm* value that specifies unblocked variable-length format records (V or VA), the *blksize* value is the *lrecl* value plus 4.
 - For a DASD device on MVS and a *recfm* value that specifies blocked records (FB, FBA, FBS, FBSA, VB, or VBA), the *blksize* value is left at 0 by Language Environment so that the system can determine the optimum *blksize* value.
 - For a terminal and a *recfm* value that specifies blocked fixed-length format records (FB, FBA, FBS, or FBSA), the *blksize* value is the same as the *lrecl* value.
 - For a terminal and a *recfm* value that specifies blocked variable-length format records (VB or VBA), the *blksize* value is the *lrecl* value plus 4.
 - For all other cases, *blksize* has a value which gives 100 records per block if the *blksize* value wouldn't exceed 32760, otherwise, a value giving the largest number of records per block such that the *blksize* value that does not exceed 32760.

Or, to put it another way:

 - For a *recfm* value that specifies blocked fixed-length format records (FB, FBA, FBS, or FBSA), the *blksize* value is $lrecl \times bfact$ where *bfact* is the largest integer not exceeding 100 such that the *blksize* value does not exceed 32760.
 - For a *recfm* value that specifies blocked variable-length format records (VB or VBA), the *blksize* value is $(lrecl \times bfact)$ plus 4 where *bfact* is the largest integer not exceeding 100 such that the *blksize* value does not exceed 32760.

- Language Environment detects certain invalid values for the MSGFILE suboptions, namely an invalid value for *recfm* and a value of *lrecl* or *blksize* that exceeds 32760. A message is printed, and any incorrect values are ignored.
- Invalid combinations of *recfm*, *lrecl*, and *blksize* values are not diagnosed by Language Environment but can cause an error condition to be detected by the system on the first attempt to write to the message file.
- Language Environment does not check the validity of the MSGFILE *ddname*. An invalid *ddname* generates an error condition on the first attempt to issue a message.
- Language Environment supports the use of a MSGFILE DDNAME dynamically allocated with the XTIOU, UCB nocapture, or DSAB-above-the-line options specified in the SVC99 parameters (S99TIOEX, S99ACUCB, S99DSABA flags).

C/C++ considerations

C `perror()` messages and output directed to `stderr` go to the MSGFILE destination.

Fortran considerations

To get the same message file function as with VS Fortran, specify `MSGFILE(FTnnF001,UA,133)` where *nn* is the unit number of the error unit. For more information, see the Fortran Run-Time Migration Guide.

PL/I considerations

Runtime messages in PL/I programs are directed to the file specified by MSGFILE, instead of to the PL/I SYSPRINT STREAM PRINT file.

User-specified output is still directed to the PL/I SYSPRINT STREAM PRINT file. To direct this output to the Language Environment MSGFILE file, specify `MSGFILE(SYSPRINT)`.

For more information

- For more information about the RPTOPTS and RPTSTG runtime options, see “RPTOPTS” on page 107 and “RPTSTG” on page 108.
- For more information about the CEEMSG and CEEMOUT callable services, see *z/OS Language Environment Programming Reference*.
- For details on how HLL compiler options affect messages, see information about HLL I/O statements and message handling in *z/OS Language Environment Programming Guide*.
- For more information about `perror()` and `stderr` see C message output information in *z/OS Language Environment Programming Guide*.
- For more information about the CESE transient data queue, see *z/OS Language Environment Programming Guide*.

MSGQ

Derivation: MeSsaGe Queue

MSGQ specifies the number of ISI blocks that Language Environment allocates on a per thread basis for use by the application. The ISI contains information for Language Environment to use when identifying and reacting to conditions, providing access to `q_data` tokens, and assigning space for message inserts used with user-created messages. When an ISI is needed and one is not available, Language Environment uses the least recently used ISI. CEECM1 allocates storage for the ISI, if necessary.

MSGQ

Non-CICS default

MSGQ=((15),OVR)

CICS default

MSGQ is ignored under CICS.

Syntax

```
▶▶ MSGQ = ( ( number ) , ( OVR | NONOVR ) ) ▶▶
```

number

An integer that specifies the number of ISIs to be maintained per thread within an enclave.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

PL/I MTF considerations

In a PL/I MTF application, MSGQ sets the number of message queues allowed for each task.

For more information

- For more information about the CEEECMI callable service, see *z/OS Language Environment Programming Reference*.
- For more information about the ISI, see *z/OS Language Environment Programming Guide*.

NATLANG

Derivation: NATional LANGuage

NATLANG specifies the initial national language to be used for the run-time environment, including error messages, month names, and day of the week names. Message translations are provided for Japanese and for uppercase and mixed-case US English. NATLANG also determines how the message facility formats messages.

NATLANG affects only the Language Environment NLS and date and time services, not the Language Environment locale callable services.

You can set the national language by using the NATLANG run-time option or the SET function of the CEE3LNG callable service. Language Environment maintains one current language at the enclave level. The current language remains in effect until it is changed. For example, if you specify JPN in the NATLANG run-time option, but later specify ENU using the CEE3LNG callable service, ENU becomes the current national language.

Non-CICS default

NATLANG=((ENU),OVR)

CICS default

NATLANG=((ENU),OVR)

Amode 64 default

NATLANG=((ENU),OVR)

Syntax

► NATLANG = (([ENU | UEN | JPN]) , [OVR | NONOVR]) ►

ENU

A 3-character ID specifying mixed-case US English.

Message text consists of SBCS characters and includes both uppercase and lowercase letters.

UEN

A 3-character ID specifying uppercase US English.

Message text consists of SBCS characters and includes only uppercase letters.

JPN

A 3-character ID specifying Japanese.

Message text can contain a mixture of SBCS and DBCS characters.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

The NATLANG option specifies the initial value for the enclave.

Usage notes

- **Restriction:** CEE3LNG and CEESLTL are not available to AMODE 64 applications.
- If you specify a national language that is not available on your system, Language Environment uses the IBM-supplied default ENU (mixed-case US English) and issues a return code of 4 and a warning message. CEEROPT, CEEUOPT, CELQROPT, and CELQUOPT can specify an unknown national language code, but give a return code of 4 and a warning message.
- Language Environment is sensitive to the national language when it writes storage reports, option reports, and dump output. When the national language is uppercase US English or Japanese, the environment variable `_CEE_UPPERCASE_DATA` can be used to determine whether variable data in storage reports, options reports and dump output is in uppercase. When this environment variable is set to YES, variable data (entry point names, program unit names, variable names, Trace Entry in EBCDIC data, and

NATLANG

hexadecimal/EBCDIC displays of storage) are changed to uppercase. When this environment variable is not set or set to a value other than YES, variable data will not be changed to uppercase.. Variable data is never changed to uppercase.when the national language is mixed case US English.

C/C++ considerations

Language Environment provides locales used in C and C++ to establish default formats for the locale-sensitive functions and locale callable services, such as date and time formatting, sorting, and currency symbols. To change the locale, you can use the `setlocale()` library function or the CEESETL callable service.

The settings of CEESETL or `setlocale()` do not affect the setting of the NATLANG runtime option. NATLANG affects only Language Environment NLS and date and time services. `setlocale()` and CEESETL affect only C/C++ locale-sensitive functions and Language Environment locale callable services.

To ensure that all settings are correct for your country, use NATLANG and either CEESETL or `setlocale()`.

PL/I MTF considerations

NATLANG affects every task in the application. The SET function of CEE3LNG is supported for the relinked OS PL/I or PL/I for MVS & VM MTF applications only.

For more information

- For more information about the CEE3LNG callable service, see *z/OS Language Environment Programming Reference*.
- For more information about `setlocale()`, see *z/OS XL C/C++ Programming Guide*.

OCSTATUS (Fortran only)

Derivation: Open Close STATUS

OCSTATUS controls the verification of file existence and whether a file is deleted based on the STATUS specifier on the OPEN and CLOSE statement, respectively.

Non-CICS default

OCSTATUS=((ON),OVR)

CICS default

OCSTATUS is ignored under CICS.

Syntax

Diagram illustrating the syntax for OCSTATUS: `OCSTATUS=((ON|OFF),OVR|NONOVR)`. The diagram shows the structure of the assignment statement with brackets indicating the options for each part of the list.

ON Specifies that file existence is checked with each OPEN statement to verify that the status of the file is consistent with STATUS='OLD' and STATUS='NEW'. It also specifies that file deletion occurs with each CLOSE statement with STATUS='DELETE' for those devices which support file deletion. Preconnected files are included in these verifications. OCSTATUS consistency checking

applies to DASD files, PDS members, VSAM files, MVS labeled tape files, and dummy files only. For dummy files, the consistency checking occurs only if the file was previously opened successfully in the current program.

In addition, when a preconnected file is disconnected by a CLOSE statement, an OPEN statement is required to reconnect the file under OCSTATUS. Following the CLOSE statement, the INQUIRE statement parameter OPENED indicates that the unit is disconnected.

OFF

Bypasses file existence checking with each OPEN statement and bypasses file deletion with each CLOSE statement.

If STATUS='NEW', a new file is created; if STATUS='OLD', the existing file is connected.

If STATUS='UNKNOWN' or 'SCRATCH', and the file exists, it is connected; if the file does not exist, a new file is created.

In addition, when a preconnected file is disconnected by a CLOSE statement, an OPEN statement is *not* required to reestablish the connection under OCSTATUS(OFF). A sequential READ, WRITE, BACKSPACE, REWIND, or ENDFILE will reconnect the file to a unit. Before the file is reconnected, the INQUIRE statement parameter OPENED will indicate that the unit is disconnected; after the connection is reestablished, the INQUIRE statement parameter OPENED will indicate that the unit is connected.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

PC (Fortran only)

Derivation: Priate Common blocks

PC controls whether Fortran static common blocks are shared among load modules.

Non-CICS default

PC=((OFF),OVR)

CICS default

PC is ignored under CICS.

Syntax

►► PC = ((OFF) , (NONOVR)) ►►

OFF

Specifies that Fortran static common blocks with the same name but in different load modules all refer to the same storage. PC(OFF) applies only to static common blocks referenced by compiled code produced by any of the following compilers and that were **not** compiled with the PC compiler option:

- VS FORTRAN Version 2 Release 5
- VS FORTRAN Version 2 Release 6

ON Specifies that Fortran static common blocks with the same name but in different load modules do not refer to the same storage.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

PLITASKCOUNT (PL/I only)

Derivation: PL/I TASK COUNTER

PLITASKCOUNT controls the maximum number of tasks active at one time while you are running PL/I MTF applications. PLITASKCOUNT(20) provides behavior compatible with the PL/I ISASIZE(,20) option.

Non-CICS default

PLITASKCOUNT=((20),OVR)

CICS default

PLITASKCOUNT is ignored under CICS.

Syntax

```
▶▶ PLITASKCOUNT = ( ( ( tasks ) , ( OVR / NONOVR ) ) ▶▶
```

tasks

A decimal integer that is the maximum number of tasks allowed in a PL/I MTF application at any one time during execution. The total tasks include the main task and subtasks created directly or indirectly from the main task.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- A value of zero (0) assumes the IBM-supplied default of 20.
- If a request to create a task would take the number of currently active tasks over the allowable limit, condition IBM0566S is signaled and the task is not created.

PL/I MTF considerations

If *tasks* or the IBM-supplied default of 20 exceeds the z/OS UNIX installation default of the maximum number of threads, Language Environment assumes the z/OS UNIX installation default.

POSIX

Derivation: Portable Operating System Interface - X

POSIX specifies whether the enclave can run with the POSIX semantics.

POSIX is an application characteristic that is maintained at the enclave level. After you have established the characteristic during enclave initialization, you cannot change it.

When you set POSIX to ON, you can use functions that are unique to POSIX, such as `pthread_create()`.

One of the effects of POSIX(ON) is the enablement of POSIX signal handling semantics, which interact closely with the Language Environment condition handling semantics.

ANSI C programs can access the z/OS UNIX file System on MVS independent of the POSIX setting. Where ambiguities exist between ANSI and POSIX semantics, the POSIX runtime options, setting indicates the POSIX semantics to follow.

Non-CICS default

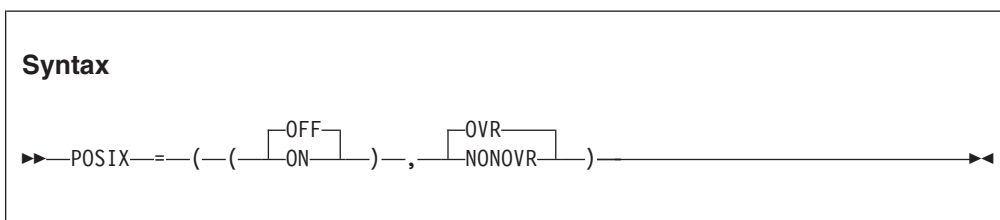
POSIX=((OFF),OVR)

CICS default

POSIX is ignored under CICS.

AMODE 64 default

POSIX=((OFF),OVR)



OFF

Indicates that the application is not POSIX-enabled.

ON

Indicates that the application is POSIX-enabled.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- If you set POSIX to ON and you run non-thread-safe languages such as PL/I and C++ in a thread other than the initial thread, the behavior is undefined.
- If you set POSIX to ON when z/OS UNIX is not active, the message file receives a warning, POSIX is set to OFF, but the application continues to run.

POSIX

- When you set POSIX to ON while an application is running under CICS, you receive a warning message, POSIX is set OFF, and the application continues to run.
- Within nested enclaves, only one enclave can have the POSIX option set to ON. All other nested enclaves must have the POSIX option set to OFF. When nested enclaves are specifying the runtime option POSIX(ON) within one Language Environment process, Language Environment will display a severity 3 error message and let abend U4039 occur with reason code 172.

For more information

- For more information about POSIX functions that have an z/OS UNIX kernel dependency, or a POSIX ON dependency (especially for a failure where the kernel dependency or the POSIX ON setting is not met), see *z/OS XL C/C++ Runtime Library Reference*.
- For more information about the INTERRUPT run-time option, see “INTERRUPT” on page 88.

PROFILE

PROFILE controls the use of an optional profiler which collects performance data for the running application.

Non-CICS default

```
PROFILE=((OFF,' '),OVR)
```

CICS default

```
PROFILE=((OFF,' '),OVR)
```

Amode 64 default

```
PROFILE=((OFF,' '),OVR)
```

Syntax

```
►► PROFILE - ( - ( OFF | ON ) , ' string ' ) - , ( OVR | NONOVR ) - ►►
```

OFF

Indicates that the profile facility is inactive.

ON Indicates that the profile facility is active.

' '

A null string indicates that no options are to be passed to the profiler.

string

Profile options that Language Environment will pass to the profiler installed. You can enclose the string in either single or double quotation marks. The maximum length of the string is 250 bytes when specified on program invocation or via a compiler directive. When specifying this option using the CEEXOPT macro, the size is limited to 242 bytes.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

For more information

An application cannot run with both the TEST and PROFILE options in effect. If both are specified, an informational message is generated and the Language Environment forces the PROFILE option OFF.

PRTUNIT (Fortran only)PRinT UNIT

Derivation: PRTUNIT identifies the unit number used for PRINT and WRITE statements that do not specify a unit number.

Non-CICS default

PRTUNIT=((6),OVR)

CICS default

PRTUNIT is ignored under CICS.

Syntax

► PRTUNIT=(number), OVR
NONOVR ◄

number

A valid unit number in the range 0-99.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

PUNUNIT (Fortran only)**Derivation:** PUNch UNIT

PUNUNIT identifies the unit number used for PUNCH statements that do not specify a unit number.

Non-CICS default

PUNUNIT=((7),OVR)

CICS default

PUNUNIT is ignored under CICS.

PUNUNIT

Syntax

►► PUNUNIT = ((*number*) , (OVR / NONOVR)) ◄◄

number

A valid unit number in the range 0-99.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

RDRUNIT (Fortran only)

Derivation: ReaDeR UNIT

RDRUNIT identifies the unit number used for READ statements that do not specify a unit number.

Non-CICS default

RDRUNIT=((5),OVR)

CICS default

RDRUNIT is ignored under CICS.

Syntax

►► RDRUNIT = ((*number*) , (OVR / NONOVR)) ◄◄

number

A valid unit number in the range 0-99.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

RECPAD (Fortran only)

Derivation: RECord PADding

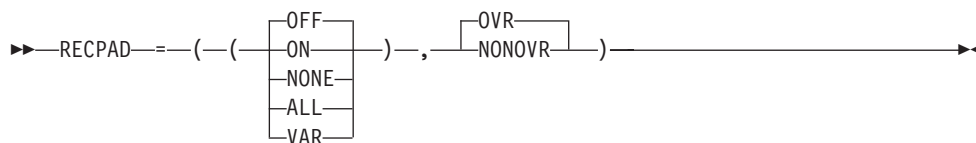
RECPAD specifies whether a formatted input record is padded with blanks.

Non-CICS default

RECPAD=((OFF),OVR)

CICS default

RECPAD is ignored under CICS.

Syntax**OFF|NONE**

Specifies that no blank padding be applied when an input list and format specification requires more data from an input record than the record contains. If more data are required, the error described by condition FOR1002 is detected.

ON|ALL

Specifies that a formatted input record within an internal file, or a varying or undefined length record (RECFM=U or V) external file, be padded with blanks when an input list and format specification require more data from the record than the record contains. Blanks added for padding are interpreted as though the input record actually contains blanks in those fields.

VAR

Applies blank padding to any of the following types of files:

- An external, non-VSAM file with a record format (the RECFM value) that allows the lengths of records to differ within the file. Such record formats are variable (V), variable blocked (VB), undefined (U), variable spanned (VS), and variable blocked spanned (VBS); this excludes fixed (F), fixed blocked (FB), and fixed blocked standard (FBS).
- An external, VSAM entry-sequenced data set (ESDS) or key-sequenced data set (KSDS).
- An internal file.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- NORECPAD has the same effect as RECPAD(OFF) and RECPAD(NONE). RECPAD has the same effect as RECPAD(ON) and RECPAD(ALL).
- The PAD specifier of the OPEN statement can be used to indicate padding for individual files.

RPTOPTS**Derivation:** RePorT OPTioNS

RPTOPTS generates, after an application has run, a report of the run-time options in effect while the application was running. RPTOPTS(ON) lists the declared run-time options in alphabetical order. The report lists the option names and

RPTOPTS

shows where each option obtained its current setting. Language Environment writes options reports only in mixed-case US English.

For an example and complete description of the options report, see *z/OS Language Environment Debugging Guide*.

Non-CICS default

RPTOPTS=((OFF),OVR)

CICS default

RPTOPTS=((OFF),OVR)

AMODE 64 default

RPTOPTS=((OFF),OVR)

Syntax

```
▶▶ RPTOPTS == ( ( OFF ON ) , ( OVR NONOVR ) ) ▶▶
```

OFF

Does not generate a report of the run-time options in effect while the application was running.

ON Generates a report of the run-time options in effect while the application was running.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- For AMODE 64 applications, Language Environment writes the options report to stderr.
- RPTOPTS may not generate the options report if your application ends abnormally.
- In a non-CICS environment, Language Environment directs the report to the *ddname* specified in the MSGFILE run-time option. Under CICS, with RPTOPTS(ON), Language Environment writes the options report to the CESE queue when the transaction ends successfully.

For more information

- See “MSGFILE” on page 94 for more information about the MSGFILE run-time option.
- For an example and complete description of the options report, see *z/OS Language Environment Debugging Guide*.

RPTSTG

Derivation: RePorT ST or aGe

RPTSTG generates, after an application has run, a report of the storage the application used. Language Environment writes storage reports only in mixed-case US English.

Use the storage report information to help you set the ANYHEAP, BELOWHEAP, HEAP, HEAP64, HEAPPOLS, HEAPPOLS64, IOHEAP64, LIBHEAP64, LIBSTACK, STACK, STACK64, THREADHEAP, THREADSTACK, and THREADSTACK64 run-time options for the best storage tuning.

For an example and complete description of the storage report, see *z/OS Language Environment Debugging Guide*.

Non-CICS default

RPTSTG=((OFF),OVR)

CICS default

RPTSTG=((OFF),OVR)

AMODE 64 default

RPTSTG=((OFF),OVR)

Syntax

► RPTSTG= ((OFF ON) , (OVR NONOVR)) ►

OFF

Does not generate a report of the storage used while the application was running.

ON Generates a report of the storage used while the application was running.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

The phrases “Number of segments allocated” and “Number of segments freed” represent, on CICS, the number of EXEC CICS GETMAIN and EXEC CICS FREEMAIN requests, respectively.

z/OS UNIX considerations

The RPTSTG option applies to storage utilization for the enclave, including thread-level information about stack utilization, and heap storage used by multiple threads.

Usage notes

- For AMODE 64 applications, Language Environment writes the storage report to stderr.

RPTSTG

- RPTSTG may not generate the storage report if your application ends abnormally.
- When a vendor heap manager (VHM) is active, the Language Environment storage report includes a text line indicating that the user heap for the C/C++ part of the enclave is managed separately. The VHM is expected to write its own storage report to the stderr stream.
- RPTSTG includes PL/I task-level information about stack and heap usage.
- The phrases “Number of segments allocated” and “Number of segments freed” represent the number of system requests to allocate and deallocate storage requests, respectively.
- If you specify the RPTSTG runtime option while using HEAPPOOLS, extra storage is obtained from the ANYHEAP and is used to complete the storage report on heappools. This extra storage is only allocated when both HEAPPOOLS and RPTSTG are used.
- If you specify the RPTSTG runtime option while using HEAPPOOLS64, extra storage is obtained from the LIBHEAP64 and is used to complete the storage report on heappools. This extra storage is only allocated when both HEAPPOOLS64 and RPTSTG are used.

Performance considerations

This option increases the time it takes for an application to run. Therefore, use it only as an aid to application development.

The storage report generated by RPTSTG(ON) shows the number of system-level calls to obtain storage that were required while the application was running. To improve performance, use the storage report numbers generated by the RPTSTG option as an aid in setting the initial and increment size for stack and heap. This reduces the number of times that the Language Environment storage manager makes requests to acquire storage. For example, you can use the storage report numbers to set appropriate values in the HEAP *init_size* and *incr_size* fields for allocating storage.

For more information

- For more information about tuning your application with storage numbers, see *z/OS Language Environment Programming Guide* or *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.
- For more information about the MSGFILE run-time option, see “MSGFILE” on page 94.
- For an example and complete description of the storage report, see *z/OS Language Environment Debugging Guide*.

RTEREUS (COBOL only)

Derivation: Run Time Environment REUSe

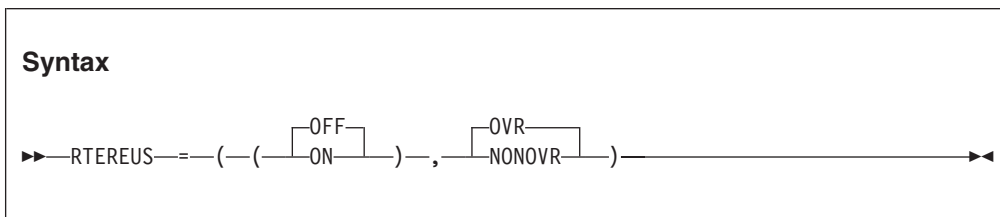
RTEREUS implicitly initializes the runtime environment to be reusable when the main program for the thread is a COBOL program. This option is valid only when specified at the system level, region level, in a CEEUOPT, or in the CEEBXITA assembler user exit.

Non-CICS default

RTEREUS=((OFF),OVR)

CICS default

RTEREUS is ignored under CICS.



OFF

Does not initialize the runtime environment to be reusable when the first COBOL program is invoked.

ON

Initializes the runtime environment to be reusable when the first COBOL program is invoked.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- Avoid using RTEREUS(ON) as a system-level or region-level default. If you do use RTEREUS, use it for specific applications only.
- **Restrictions:**
 - RTEREUS(ON) cannot be used with XPLINK(ON).
 - RTEREUS(ON) cannot be used in a z/OS UNIX process.
 - Enterprise COBOL programs compiled with the THREAD compiler option do not run with RTEREUS(ON).
- Under Language Environment, RTEREUS(ON) is only supported in a single enclave environment unless you modify the behavior using the IGZERREO CSECT. With the IBM-supplied default setting for COBOL's reusable environment, applications that attempt to create nested enclaves terminate with error message IGZ0168S.

Nested enclaves can be created by applications that use SVC LINK or CMSCALL to invoke application programs. One example is when an SVC LINK is used to invoke an application program under ISPF that is using ISPF services (such as CALL 'ISPLINK' and ISPF SELECT).
- If a Language Environment reusable environment is established (using RTEREUS), attempts to run a C or PL/I main program under Language Environment will fail. For example, when running on ISPF with RTEREUS(ON):
 - The first program invoked by ISPF is a COBOL program. A Language Environment reusable environment is established.
 - At some other point, ISPF invokes a PL/I or C program. The initialization of the PL/I or C program will fail.
- If a large number of COBOL programs are run (using RTEREUS) under the same MVS task, you can encounter out-of-region abends. This is because all storage acquired by Language Environment to run COBOL programs is kept in storage until the MVS task ends or the Language Environment environment is terminated.

RTEREUS

- Language Environment storage and runtime options reports are not produced by Language Environment (using RTEREUS) unless a STOP RUN is issued to end the enclave.
- The IGZERREO CSECT affects the handling of program checks in the non-Language Environment-enabled driver that repeatedly invokes COBOL programs. It also affects the behavior of running COBOL programs in a nested enclave when a reusable environment is active.

IMS considerations

RTEREUS is not recommended for use under IMS.

Performance considerations

You must change STOP RUN statements to GOBACK statements in order to gain the benefits of RTEREUS. STOP RUN terminates the reusable environment. If you specify RTEREUS and use STOP RUN, Language Environment recreates the reusable environment on the next invocation of COBOL. Doing this repeatedly degrades performance, because a reusable environment takes longer to create than does a normal environment.

The IGZERREO CSECT affects the performance of running with RTEREUS.

Language Environment also offers preinitialization support in addition to RTEREUS.

For more information

- For more information about specifying this option at the system or region level, see Chapter 5, “Customizing Language Environment runtime options,” on page 19.
- For more information about preinitialization and specifying this option in a CEEUOPT, see *z/OS Language Environment Programming Guide*.
- For more information about IGZERREO, see “Modifying COBOL reusable environment behavior” on page 174.

SIMVRD (COBOL only)

Derivation: SIMulate Variable length Relative organization Data sets

SIMVRD specifies whether your COBOL programs use a VSAM KSDS to simulate variable-length relative organization data sets.

Non-CICS default

SIMVRD=((OFF),OVR)

CICS default

SIMVRD is ignored under CICS.

Syntax

► SIMVRD = ((OFF) , (OVR))

OFF

Do not use a VSAM KSDS to simulate variable-length relative organization.

ON Use a VSAM KSDS to simulate variable-length relative organization.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

For more information

See the appropriate version of the programming guide in the COBOL library at Enterprise COBOL for z/OS library (<http://www-01.ibm.com/support/docview.wss?uid=swg27036733>).

STACK

STACK controls the allocation of the thread's stack storage for both the upward and downward-growing stacks. Typical items residing in the upward-growing stack are C or PL/I automatic variables, COBOL LOCAL-STORAGE data items, and work areas for COBOL library routines.

The downward growing stack is allocated only when an application has been built with XPLINK.

Storage required for the common anchor area (CAA) and other control blocks is allocated separately from, and before, the allocation of the initial stack segment and the initial heap.

Non-CICS default

STACK=((128K,128K,ANYWHERE,KEEP,512K,128K),OVR)

CICS default

STACK=((4K,4080,ANYWHERE,KEEP,4K,4080),OVR)

Syntax

```

▶▶ STACK = ( ( ( -usinit_size-, -usincr_size-, [ ANYWHERE | ANY | BELOW ],
[ KEEP | FREE ], -dsinit_size-, -dsincr_size- )-, [ OVR | NONOVR ] ) )

```

usinit_size

Determines the size of the initial upward-growing stack segment. The storage is contiguous. You specify the *usinit_size* value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

usinit_size can be preceded by a minus sign. In environments other than CICS, if you specify a negative number Language Environment uses all available storage minus the amount specified for the initial stack segment.

STACK

A size of "0" or "-0" requests half of the largest block of contiguous storage in the region below the 16-MB line.

usincr_size

Determines the minimum size of any subsequent increment to the upward-growing stack area. You can specify this value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is the larger of two values (*usincr_size* or the requested size) rounded up to the nearest multiple of 8 bytes

If you specify *usincr_size* as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 8 bytes, is obtained.

The requested size is the amount of storage a routine needs for a stack frame. For example, if the requested size is 9000 bytes, *usincr_size* is specified as 8K, and the initial stack segment is full, Language Environment gets a 9000 byte stack increment from the operating system to satisfy the request. If the requested size is smaller than 8K, Language Environment gets an 8K stack increment from the operating system.

ANYWHERE | ANY | BELOW

Specifies the storage location. For downward growing stack, this option is ignored and the storage is always placed above 16 MB.

BELOW

Specifies that the stack storage must be allocated below the 16MB line in storage that is accessible to 24-bit addressing.

ANYWHERE | ANY

Specifies that stack storage can be allocated anywhere in storage. If there is no storage available above the line, Language Environment acquires storage below the 16-MB line.

KEEP | FREE

Determines the disposition of the storage increments when the last stack frame in the increment segment is freed.

KEEP

Specifies that storage allocated to stack increments is not released when the last of the storage in the stack increment is freed.

FREE

Specifies that storage allocated to stack increments is released when the last of the storage in the stack is freed. The initial stack segment is never released until the enclave terminates.

dsinit_size

Determines the size of the initial downward growing stack segment. The storage is contiguous. You specify the *dsinit_size* value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 16 bytes.

dsincr_size

Determines the minimum size of any subsequent increment to the downward growing stack area. You can specify this value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is the larger of two values-- *dsincr_size* or the requested size--rounded up to the nearest multiple of 16 bytes.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

dsinit_size and *dsincr_size* suboptions are ignored under CICS.

The maximum initial and increment size for CICS above 16 MB is 1 gigabyte (1024 MB). This restriction is subject to change from one release of CICS to another.

Both the initial size and the increment size are rounded up to the nearest multiple of 8 bytes. The initial size minimum is 4 KB.

If you do not specify STACK, Language Environment assumes the default value of 4 KB. Under CICS, STACK(0), STACK (-0), and STACK (-n) are all interpreted as STACK(4K).

The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16 bytes CICS storage check zone. Without this accommodation, an extra page of storage is allocated (only when the storage allocation is below the 16MB line).

z/OS UNIX considerations

The STACK option specifies the characteristics of the user stack for the initial thread. In particular, it gets the initial size of the user stack for the initial thread.

The characteristics that indicate *incr_size*, ANYWHERE, and KEEP | FREE apply to any thread created using *pthread_create*. Language Environment gets the initial stack size from the thread's attribute object specified in the *pthread_create* function. The default size to be set in the thread's attribute object is obtained from the STACK run-time option's initial size.

The recommended default setting for STACK under z/OS UNIX is STACK=((12K,12K,ANYWHERE,KEEP,512K,128K),OVR).

Usage notes

When an application is running in an XPLINK environment (that is, either the XPLINK(ON) runtime option was specified, or the initial program contained at least one XPLINK-compiled part), the STACK runtime option will be forced to STACK(,,ANY,,,). Only the third suboption of the STACK runtime option is changed by this action, to indicate that stack storage can be allocated anywhere in storage. No message will be issued to indicate this action. In this case, if a Language Environment runtime options report is generated using the RPTOPTS runtime option, the STACK option will be reported as "Override" under the LAST WHERE SET column.

The *dsinit_size* and *dsincr_size* values are the amounts of storage that can be used for downward growing stack frames (plus the stack header, approximately 20 bytes). The actual size of the storage getmained will be 4K (8K if a 4K page alignment cannot be guaranteed) larger to accommodate the guard area.

The downward growing stack is only initialized in an XPLINK supported environment, and only when an XPLINK application is active in the enclave. Otherwise the suboptions for the downward growing stack are ignored.

Applications running with ALL31(OFF) must specify STACK(,,BELOW,,) to ensure that stack storage is addressable by the application.

STACK

PL/I considerations

PL/I automatic storage above the 16-MB line is supported under control of the Language Environment STACK option. When the Language Environment stack is above, PL/I temporaries (dummy arguments) and parameter lists (for reentrant/recursive blocks) also reside above.

The stack frame size for an individual block is constrained to 16MB. Stack frame extensions are also constrained to 16MB. Therefore, the size of an automatic aggregate, temporary variable, or dummy argument cannot exceed 16MB. Violation of this constraint might have unpredictable results.

If an OS PL/I application does not contain any edited stream I/O and if it is running with AMODE 31, you can relink it with Language Environment to use STACK(,ANY,,). Doing so is particularly useful under CICS to help relieve below-the-line storage constraints.

PL/I MTF considerations

The STACK option allocates and manages stack storage for the PL/I main task only. For information about stack storage management in the subtasks, see "THREADSTACK" on page 131.

Performance considerations

To improve performance, use the storage report numbers generated by the RPTSTG run-time option as an aid in setting the initial and increment sizes for STACK.

For more information

- See "ALL31" on page 50, for more information about the ALL31 run-time option.
- See "RPTSTG" on page 108, for more information about the RPTSTG run-time option.
- See "THREADSTACK" on page 131, for more information about the THREADSTACK run-time option.
- For more information about using the storage reports generated by the RPTSTG run-time option to tune the stacks, see *z/OS Language Environment Programming Guide*.

STACK64 (AMODE 64 only)

Derivation: STACK storage for AMODE 64

STACK64 controls the allocation of the thread's stack storage for AMODE 64 applications.

AMODE 64 default

STACK64=((1M,1M,128M),OVR)

Syntax

```

▶▶ STACK64 == ( ( initial , increment , maximum ) , OVR  
NONOVR )
▶ )

```

initial

Determines the size of the initial stack segment. The storage is contiguous. This value is specified as *nM* bytes of storage.

increment

Determines the minimum size of any subsequent increment to the downward-growing stack area. This value is specified as *nM* bytes of storage. The actual amount of allocated storage is the larger of two values— *increment* or the requested size—rounded up to the nearest 1MB.

If you specify *increment* as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 1MB, is obtained.

The requested size is the amount of storage a routine needs for a stack frame.

maximum

Specifies the maximum stack size. This value is specified as *nM* bytes of storage. When the maximum size is less than the initial size, *initial* is used as the maximum stack size.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- The 1 MB guard area is not included in any of the sizes.
- The maximum stack segment is the maximum of STACK64 initial and maximum sizes.
- When a multithreaded application that creates many pthreads is run, the default value of 128MB for the maximum stack size of the STACK64 and THREADSTACK64 runtime options might cause excessive use of system resources, such as real storage. For such applications, you need to use the Language Environment Storage Report (RPTSTG runtime option) to determine the actual pthread stack storage usage of your application, and then use the THREADSTACK64 runtime option to set the maximum stack size to a value closer to the actual usage.

Performance considerations

To improve performance, use the storage report numbers generated by the RPTSTG runtime option as an aid in setting the initial and increment sizes for STACK64.

For more information

- See “RPTSTG” on page 108 for more information about the RPTSTG runtime option.

- For more information about using the storage reports generated by the RPTSTG runtime option to tune the stacks for AMODE 64 applications, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

STORAGE

STORAGE controls the initial content of storage when allocated and freed. It also controls the amount of storage that is reserved for the out-of-storage condition. If you specify one of the parameters in the STORAGE run-time option, all allocated storage processed by that parameter is initialized to the specified value. Otherwise, it is left uninitialized.

You can use the STORAGE option to identify uninitialized application variables, or prevent the accidental use of previously freed storage. STORAGE is also useful in data security. For example, storage containing sensitive data can be cleared when it is freed.

Non-CICS default

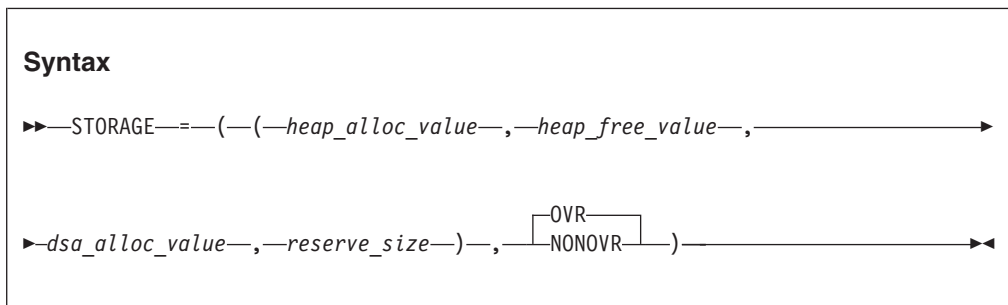
STORAGE=((NONE,NONE,NONE,0K),OVR)

CICS default

STORAGE=((NONE,NONE,NONE,0K),OVR)

Amode 64 default

STORAGE=((NONE,NONE,NONE,),OVR)



heap_alloc_value

The initialized value of any heap storage allocated by the storage manager. You can specify *heap_alloc_value* as:

- A single character enclosed in quotation marks. If you specify a single character, every byte of heap storage allocated by the storage manager is initialized to that character's EBCDIC equivalent. For example, if you specify 'a' as the *heap_alloc_value*, heap storage is initialized to X'818181...81' or 'aaa...a'.
- Two hex digits without quotation marks. If you specify two hex digits, every byte of the allocated heap storage is initialized to that value. For example, if you specify FE as the *heap_alloc_value*, heap storage is initialized to X'FEFEFE...FE'. A *heap_alloc_value* of 00 initializes heap storage to X'0000...00'.
- **NONE**. If you specify NONE, the allocated heap storage is not initialized.

heap_free_value

The value of any heap storage freed by the storage manager is overwritten. You can specify *heap_free_value* as:

- A single character enclosed in quotation marks. For example, a *heap_free_value* of 'f' overwrites freed heap storage to X'868686...86'; 'B' overwrites freed heap storage to X'C2'.
- Two hex digits without quotation marks. A *heap_free_value* of FE overwrites freed heap storage with X'FEFEFE...FE'.
- **NONE**. If you specify NONE, the freed heap storage is not initialized.

dsa_alloc_value

The initialized value of stack frames from the Language Environment stack. A stack frame is dynamically acquired storage that is composed of a standard register save area and the area available for automatic storage.

If specified, all Language Environment stack storage, including automatic variable storage, is initialized to *dsa_alloc_value*. Stack frames allocated outside the Language Environment stack are never initialized.

You can specify *dsa_alloc_value* as:

- A single character enclosed in quotation marks. If you specify a single character, any dynamically acquired stack storage allocated by the storage manager is initialized to that character's EBCDIC equivalent. For example, if you specify 'A' as the *dsa_alloc_value*, stack storage is initialized to X'C1'. A *dsa_alloc_value* of 'F' initializes stack storage to X'C6', 'd' to X'84'.
- Two hex digits without quotation marks. If you specify two hex digits, any dynamically acquired stack storage is initialized to that value. For example, if you specify FE as the *dsa_alloc_value*, stack storage is initialized to X'FE'. A *dsa_alloc_value* of 00 initializes stack storage to X'00', FF to X'FF'.
- **CLEAR** If you specify CLEAR, any unused portion of the initial upward growing stack segment is initialized to binary zeros, just before the main procedure gains control. This value has no effect on any stack increments or on the XPLINK or AMODE 64 downward growing stack.
- **NONE**. If you specify NONE, the stack storage is not initialized.

reserve_size

The amount of storage for the Language Environment storage manager to reserve in the event of an out-of-storage condition. You can specify the *reserve_size* value as *n*, *nK*, or *nM* bytes of storage. The amount of storage is rounded to the nearest multiple of 8 bytes.

Restriction: This suboption is ignored for AMODE 64 applications.

The default *reserve_size* is 0, so no reserve segment is allocated. If you do not specify a reserve segment and your application exhausts storage, the application terminates with abend 4088 and a reason code of 1024.

If you specify *reserve_size* as 0, no reserve segment is allocated. If you do not specify a reserve segment and your application exhausts storage, the application terminates with abend 4088 and a reason code of 1004.

If you specify a *reserve_size* that is greater than 0 on a non-CICS system, Language Environment does not immediately abend when your application runs out of storage. Instead, when the stack overflows, Language Environment uses the reserve stack as the new segment and signals a CEE0PD out of storage condition. This allows a user-written condition handler to gain control for this signal and release storage. If the reserve stack segment overflows while this is happening, Language Environment terminates with abend 4088 and reason code of 1004. The reserve stack segment is not freed until thread termination. It is acquired from 31-bit storage if the STACK(„ANY,„) runtime option is set or 24-bit storage when STACK(„BELOW,„) is requested. If a

STORAGE

determination is made to activate the reserve stack, the reserve size should be set to a minimum of 32 KB to support Language Environment condition handling and messaging internal routines as well as the user condition handler. When the reserve stack is used in a multithreaded environment, it is suggested that the ALL31(ON) and STACK(„ANY,,,) options also be in effect.

If unsuccessful, Language Environment temporarily adds the reserve stack segment to the overflowing stack, and signals the out-of-storage condition. This causes a user-written condition handler to gain control and release storage. If the reserve stack segment overflows while this is happening, Language Environment terminates with abend 4088 and reason code of 1004.

To avoid such an overflow, increase the size of the reserve stack segment with the STORAGE(„,reserve_size) run-time option. The reserve stack segment is not freed until thread termination.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

- The out-of-storage condition is not raised under CICS. Therefore, the reserve stack size (fourth suboption) is ignored under CICS and displays as 0 in all options reports

z/OS UNIX considerations

A reserve stack of the size specified by the *reserve_size* suboption of STORAGE is allocated for each thread.

Usage notes

- The behavior of the *dsa_alloc_value* suboption of the STORAGE runtime option is different for an XPLINK stack. The DSA is only initialized for routines that perform an explicit check for stack overflow. (For C/C++, the compiler option XPLINK(NOGUARD) can be used to force the compiler to generate prologs with explicit checks for stack overflow.)
- *heap_alloc_value*, *heap_free_value*, and *dsa_alloc_value* can all be enclosed in quotation marks. To initialize heap storage to the EBCDIC equivalent of a single quotation mark, double it within the string delimited by single quotation mark or surround it with a pair of double quotation marks. Both of the following examples are correct ways to specify a single quotation mark:

```
STORAGE('''')
STORAGE("''")
```

Similarly, double quotation marks must be doubled within a string delimited by double quotation marks, or surrounded by a pair of single quotation marks. The following are correct ways to specify a double quotation mark:

```
STORAGE('""')
STORAGE('""')
```

- CLEAR is not a valid option for AMODE 64 applications.
- If the initial stack segment is too small to contain the main procedure, it is allocated from the stack increment, and is not be cleared even if the CLEAR option is specified.
- If you specify CLEAR, any unused portion of the initial upward growing stack segment is initialized to binary zeros, just before the main procedure gains

control. If a small initial stack segment size is specified, the DSA of the main procedure can be allocated in the stack increment, not in the initial stack segment. In this case, the variables of the main procedure cannot be initialized to binary zeros because they are in the stack increment, not the initial stack. To prevent this, the size of the initial stack segment needs to be increased.

COBOL considerations

If you are using WSCLEAR in VS COBOL II, STORAGE(00,NONE,NONE,0K) is recommended.

Performance considerations

Using STORAGE to control initial values can increase program runtime. If you specify a *dsa_alloc_value*, performance is likely to be poor. Therefore, use the *dsa_alloc_value* option only for debugging, not to initialize automatic variables or data structures. You should not use STORAGE(,00) in any performance-critical application.

Use STORAGE(NONE,NONE,NONE,0K) when you are not debugging.

TERMTHDACT

Derivation: TERminating THreaD ACTions

TERMTHDACT sets the level of information that is produced when Language Environment percolates a condition of severity 2 or greater beyond the first routine's stack frame.

The Language Environment service CEE3DMP is called for TRACE, UATRACE, DUMP and UADUMP suboptions of TERMTHDACT.

The following CEE3DMP options are used for TRACE and UATRACE:

```
| NOBLOCKS CONDITION ENCLAVE(ALL) NOENTRY FILES
| FNAME(CEEDUMP) GENOPTS STACKFRAME(ALL) NOSTORAGE
| THREAD(ALL) TRACEBACK VARIABLES
```

The following options are used for DUMP and UADUMP:

```
| BLOCKS CONDITION ENCLAVE(ALL) NOENTRY FILES FNAME(CEEDUMP)
| GENOPTS STACKFRAME(ALL) STORAGE THREAD(ALL) TRACEBACK
| VARIABLES
```

If a message is printed, based upon the TERMTHDACT(MSG) run-time option, the message is for the active condition immediately before the termination imminent step. In addition, if that active condition is a promoted condition (was not the original condition), the original condition's message is printed.

If the TRACE runtime option is specified with the DUMP suboption, a dump containing the trace table, at a minimum, is produced. The contents of the dump depend on the values set in the TERMTHDACT runtime option.

Under normal termination, the following dump contents are generated:

Independent of the TERMTHDACT setting, Language Environment generates a dump containing the trace table only.

Non-CICS default

TERMTHDACT=((TRACE,CESE,96),OVR)

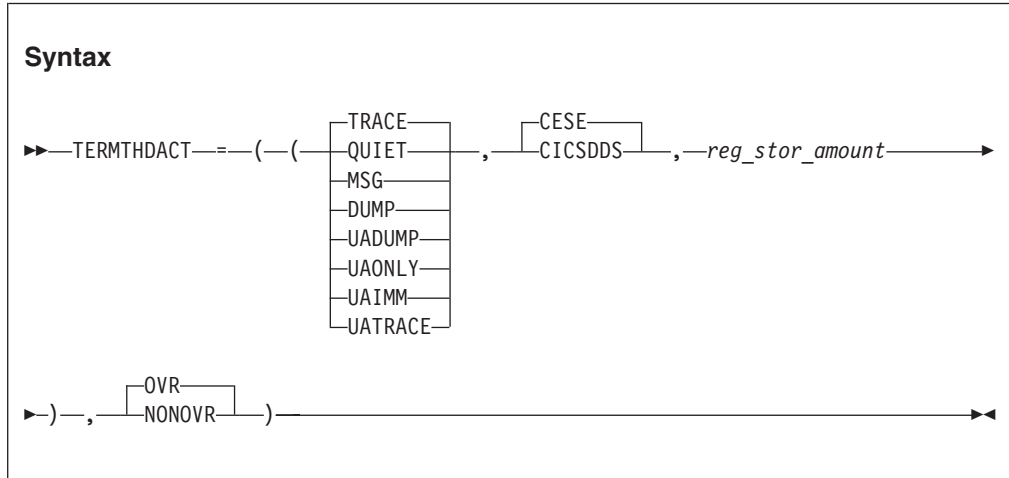
TERMTHDACT

CICS default

TERMTHDACT=((TRACE,CESE,96),OVR)

AMODE 64 default

TERMTHDACT=(TRACE,,96),OVR)



TRACE

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination and a trace of the active routines on the activation stack.

QUIET

Specifies that Language Environment does not generate a message when a thread terminates due to an unhandled condition of severity 2 or greater.

MSG

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination.

DUMP

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination, a trace of the active routines on the activation stack, and a Language Environment dump.

UADUMP

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination, a Language Environment dump, and generates a U4039 abend which allows a system dump of the user address space to be generated. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space. Under CICS, you will get a CICS transaction dump.

UATRACE

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination, a trace of the active routines on the activation stack, and generates a U4039 abend which allows a system dump of the user address space to be generated. Under non-CICS, if the appropriate DD

statement is used, you will get a system dump of your user address space. Under CICS, you will get a CICS transaction dump.

UAONLY

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space. Under CICS, you will get a CICS transaction dump.

UAIMM

Specifies to Language Environment that prior to condition management processing, for abends and program interrupts that are conditions of Severity 2 or higher, Language Environment will immediately request the operating system to generate a system dump of the original abend/program interrupt of the user address space. Due to an unhandled condition of severity 2 or greater, Language Environment generates a U4039 abend which allows a system dump of the user address space to be generated. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space. After the dump is taken by the operating system, Language Environment condition manager can continue processing. If the thread terminates due to an unhandled condition of Severity 2 or higher, then Language Environment will terminate as if TERMTHDACT(QUIET) was specified.

Note: For software-raised conditions or signals, UAIMM behaves the same as UAONLY. When TRAP(ON,SPIE) is in effect, UAIMM will yield UAONLY behavior.

CESE

Restriction: This suboption is ignored for AMODE 64 applications. Specifies that Language Environment dump output will be written to the CESE queue.

CICSDDS

Restriction: This suboption is ignored for AMODE 64 applications.

Specifies that Language Environment dump output will be written to the CICS transaction dump data set that contains both CICS and CEEDUMP data. For program checks or ABENDs, the CICSDDS option directs Language Environment to place the message output in the CICS dump dataset created for the failure. For software-raised errors, like subscript range exceeded, the CESE queue remains the destination for the output (since there may be no transaction dump for these). CICSDDS can be specified with any of the first TERMTHDACT settings except DUMP and UADUMP. Attempts to request this combination will result in an error in building the options module.

reg_stor_amount

Controls the amount of storage to be dumped around registers. This amount can be in the range from 0 to 256 bytes. The amount specified is rounded up to the nearest multiple of 32. The default amount is 96 bytes.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

- All TERMTHDACT output is written to the data queue based on the setting of CESE or CICSDDS.

See Table 9 for help in understanding the results of the different options that are available.

Table 9. Condition handling of 0Cx ABENDS in a CICS environment

<i>Options</i>	TERMTHDACT(option,CESE,)	TERMTHDACT(option,CICSDDS,)
QUIET	<ul style="list-style-type: none"> • No output. • ASRA or user ABEND issued. 	<ul style="list-style-type: none"> • No output. • ASRA or user ABEND issued.
MSG	<ul style="list-style-type: none"> • Message written to CESE queue or MSGFILE. • ASRA or user ABEND issued. 	<ul style="list-style-type: none"> • Message written to CESE queue or MSGFILE. • ASRA or user ABEND issued.
TRACE	<ul style="list-style-type: none"> • Message written to CESE queue. • Traceback written to CESE queue. • ASRA or user ABEND issued. 	<ul style="list-style-type: none"> • Message written to CESE or MSGFILE. • Traceback included in CICS transaction dump for this ABEND. • ASRA or user ABEND issued.
DUMP	<ul style="list-style-type: none"> • Message written to CESE queue. • Traceback written to CESE queue. • CEEDUMP to CESE queue. • ASRA or user ABEND issued. 	<ul style="list-style-type: none"> • Incorrect suboption combination. Not supported.
UATRACE	<ul style="list-style-type: none"> • Message written to CESE queue. • Traceback included in CICS transaction dump for this ABEND. • U4039 transaction dump in CICS dump data set. • ASRA or user ABEND issued. 	<ul style="list-style-type: none"> • Message written to CESE queue. • Traceback written to CESE queue. • U4039 transaction dump in CICS dump data set. • ASRA or user ABEND issued.
UADUMP	<ul style="list-style-type: none"> • Message written to CESE queue. • Traceback written to CESE queue. • CEEDUMP written to CESE queue. • U4039 transaction dump in CICS dump data set. • ASRA or user ABEND issued. 	<ul style="list-style-type: none"> • Incorrect suboption combination. Not supported.
UAONLY	<ul style="list-style-type: none"> • U4039 transaction dump in CICS dump data set. 	<ul style="list-style-type: none"> • No changes in behavior for CICSDDS.
UAIMM	<ul style="list-style-type: none"> • U4039 transaction dump in CICS dump data set. 	<ul style="list-style-type: none"> • No changes in behavior for CICSDDS.

Note: Program checks end in ASRx (most commonly ASRA) CICS abend with a CICS dump in the dump data set. Abends end with the abend code provided on the EXEC CICS ABEND command with a CICS dump in the dump data set if the NODUMP option was NOT specified.

For software raised errors of severity 2 or higher in a CICS environment:

Table 10. Handling of software-raised conditions in a CICS environment

Options	TERMTHDACT(option,CESE,)	TERMTHDACT(option,CICSDDS,)
QUIET	<ul style="list-style-type: none"> No output. U4038 abend issued with CANCEL and NODUMP options. 	<ul style="list-style-type: none"> No output. U4038 abend issued with CANCEL and NODUMP options.
MSG	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. U4038 abend issued. 	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. U4038 abend issued.
TRACE	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. Traceback written to CESE queue. U4038 abend issued. 	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. Traceback written to CESE queue. U4038 abend issued.
DUMP	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. Traceback written to CESE queue. CEEDUMP written to CESE queue. U4038 abend issued. 	<ul style="list-style-type: none"> Invalid suboption combination. Not supported.
UATRACE	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. Traceback written to CESE queue. U4039 transaction dump in CICS dump data set. U4038 abend issued. 	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. Traceback written to CESE queue. U4039 transaction dump in CICS dump data set. U4038 abend issued.
UADUMP	<ul style="list-style-type: none"> Message written to CESE queue or MSGFILE. Traceback written to CESE queue. CEEDUMP written to CESE queue. U4039 transaction dump in CICS dump data set. U4038 abend issued. 	<ul style="list-style-type: none"> Invalid suboption combination. Not supported.
UAONLY	<ul style="list-style-type: none"> U4039 transaction dump in CICS dump data set. U4038 abend issued. 	<ul style="list-style-type: none"> No changes in behavior for CICSDDS.
UAIMM	<ul style="list-style-type: none"> U4039 transaction dump in CICS dump data set. U4038 abend issued. 	<ul style="list-style-type: none"> No changes in behavior for CICSDDS.

Note:

1. See *z/OS Language Environment Runtime Messages* for more complete details regarding the U4039 abend.
2. When assembling a CEEROPT or CEEUOPT, the CICSDDS option cannot be issued with DUMP, or UADUMP. This results in an RC=8, CEEXOPT issues and MNOTE, and the setting is forced to TRACE.
3. Running with something like TERMTHDACT(TRACE,CICSDDS) in the CEECOPT group or CEEROPT and then creating a CEEUOPT without

TERMTHDACT

specifying the second operand (for example, TERMTHDACT(DUMP)) results in the CICS dump data set as the output destination and the following message occurs in the CESE queue:

```
CEE3627I The following messages pertain to the programmer default
runtime options.
CEE3775W A conflict was detected between the TERMTHDACT suboptions
CICSDDS and DUMP.
The TERMTHDACT level setting has been set to TRACE.
```

and the traceback is written to the CICS transaction dump data set.

z/OS UNIX considerations

The TERMTHDACT option applies when a thread terminates abnormally. Abnormal termination of a single thread causes termination of the entire enclave. If an unhandled condition of severity 2 or higher percolates beyond the first routine's stack frame, the enclave terminates abnormally.

If an enclave terminates due to a POSIX default signal action, TERMTHDACT applies only to conditions that result from program checks or abends.

Usage notes

A runtime options report will be generated and placed at the end of the enclave information whenever the TRACE, UATRACE, DUMP and UADUMP options are invoked.

COBOL considerations

TERMTHDACT(UADUMP) produces debugging information that is similar to the information produced by previous levels of COBOL.

PL/I considerations

After a normal return from a PL/I ERROR ON-unit or from a PL/I FINISH ON-unit, Language Environment considers the condition unhandled. If a GOTO is not performed and the resume cursor is not moved, the thread terminates. The TERMTHDACT setting guides the amount of information that is produced. The message is not presented twice.

PL/I MTF considerations

- TERMTHDACT applies to a task when the task terminates abnormally due to an unhandled condition of severity 2 or higher that is percolated beyond the initial routine's stack frame.
- When a task ends with a normal return from an ERROR ON-unit and other tasks are still active, a dump is not produced even when the TERMTHDACT option DUMP, UADUMP, UAONLY, or UAIMM is specified.
- All active subtasks created from the incurring task also terminate abnormally, but the enclave can continue to run.

For more information

- See "TRACE" on page 135, for more information about the TRACE runtime option.
- For more information about the CEE3DMP service and its parameters, see *z/OS Language Environment Programming Reference*.
- See *z/OS Language Environment Programming Guide* for more information about the TERMTHDACT run-time option and condition message.

- For more information about CESE, see *z/OS Language Environment Programming Guide*.

TEST | NOTEST

TEST specifies the conditions under which a debug tool (such as the Debug Tool supplied with z/OS) assumes control when the user application is being initialized. Parameters of the TEST and NOTEST run-time options are merged as one set of parameters.

Non-CICS default

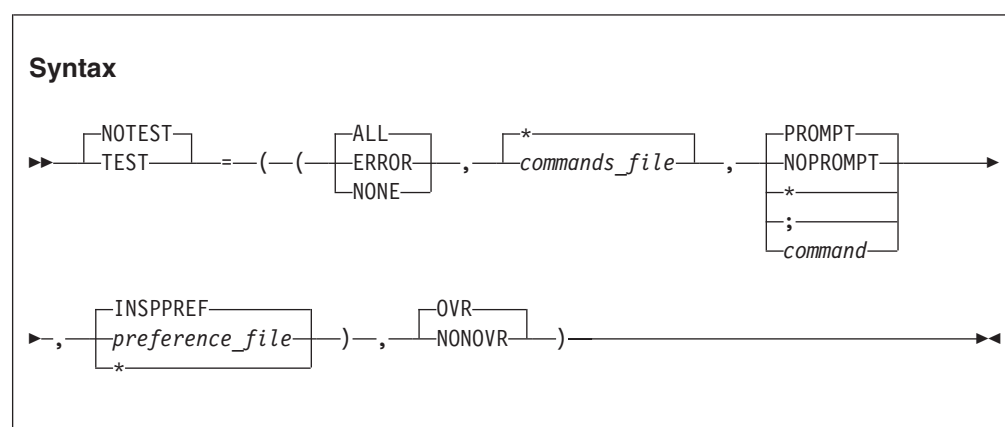
NOTEST=((ALL,*,PROMPT,INSPPREF),OVR)

CICS default

NOTEST=((ALL,*,PROMPT,INSPPREF),OVR)

Amode 64 default

NOTEST=((ALL,*,PROMPT,INSPPREF),OVR)



ALL

Specifies that any of the following causes the debug tool to gain control even without a defined AT OCCURRENCE for a particular condition or AT TERMINATION:

- The ATTENTION function
- Any Language Environment condition of severity 1 or above
- Application termination

ERROR

Specifies that only one of the following causes the debug tool to gain control without a defined AT OCCURRENCE for a particular condition or AT TERMINATION:

- The ATTENTION function
- Any Language Environment-defined error condition of severity 2 or higher
- Application termination

NONE

Specifies that no condition causes the debug tool to gain control without a defined AT OCCURRENCE for a particular condition or AT TERMINATION.

commands_file

A valid *ddname*, data set name (MVS), or file name (CMS), specifying the primary commands file for this run. If you do not specify this parameter all requests for commands go to the user terminal.

You can enclose *commands_file* in single or double quotation marks to distinguish it from the rest of the TEST | NOTEST suboption list. It can have a maximum length of 80 characters. If the data set name provided could be interpreted as a *ddname*, it must be preceded by a slash (/). The slash and data set name must be enclosed in quotation marks.

A primary commands file is required when running in a batch environment.

*** (asterisk – in place of *commands_file*)**

Specifies that no *commands_file* is supplied. The terminal, if available, is used as the source of the debug tool commands.

PROMPT

Specifies that the debug tool is invoked at Language Environment initialization.

NOPROMPT

Specifies that the debug tool is not invoked at Language Environment initialization.

*** (asterisk – in place of PROMPT/NOPROMPT)**

Specifies that the debug tool is not invoked at Language Environment initialization; equivalent to NOPROMPT.

; (semicolon – in place of PROMPT/NOPROMPT)

Specifies that the debug tool is invoked at Language Environment initialization; equivalent to PROMPT.

command

A character string that specifies a valid debug tool command. The command list can be enclosed in single or double quotation marks to distinguish it from the rest of the TEST parameter list; it cannot contain DBCS characters. Quotation marks are needed whenever the command list contains embedded blanks, commas, semicolons, or parentheses. The list can have a maximum of 250 characters.

preference_file

A valid *ddname*, data set name (MVS), or file name (CMS), specifying the preference file to be used. A preference file is a type of commands file that you can use to specify settings for your debugging environment. It is analogous to creating a profile for a text editor, or initializing an S/370 terminal session.

You can enclose *preference_file* in single or double quotation marks to distinguish it from the rest of the TEST parameter list. It can have a maximum of 80 characters.

If a specified data set name could be interpreted as a *ddname*, it must be preceded by a slash (/). The slash and data set name must be enclosed in quotation marks.

The IBM-supplied default setting for *preference_file* is INSPREF.

*** (asterisk – in place of *preference_file*)**

Specifies that no *preference_file* is supplied.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

Language Environment honors the initial command string before the main routine runs on the initial thread.

The test level (ALL, ERROR, NONE) applies to the enclave.

Language Environment honors the preference file when the debug tool is initialized, regardless of which thread first requests the debug tool services.

Usage notes

- You can specify parameters on the NOTEST option. If NOTEST is in effect when the application gains control, it is interpreted as TEST(NONE,,*). If Debug Tool is initialized using a CALL CEETEST or equivalent, the initial test level, the initial *commands_file*, and the initial *preference_file* are taken from the NOTEST runtime setting.

Performance considerations

To improve performance, use this option only while debugging.

For more information

See Debug Tool publications for details and examples of the TEST run-time option as it relates to Debug Tool.

THREADHEAP

Derivation: THREAD level HEAP storage

THREADHEAP controls the allocation and management of thread-level heap storage. Separate heap segments are allocated and freed for each thread based on the THREADHEAP specification.

For PL/I MTF applications, controlled and based variables declared in a subtask are allocated from heap storage specified by THREADHEAP. Variables in the main task are allocated from heap storage specified by HEAP.

Library use of heap storage in a substack is allocated from the enclave-level heap storage specified by the ANYHEAP and BELOWHEAP options.

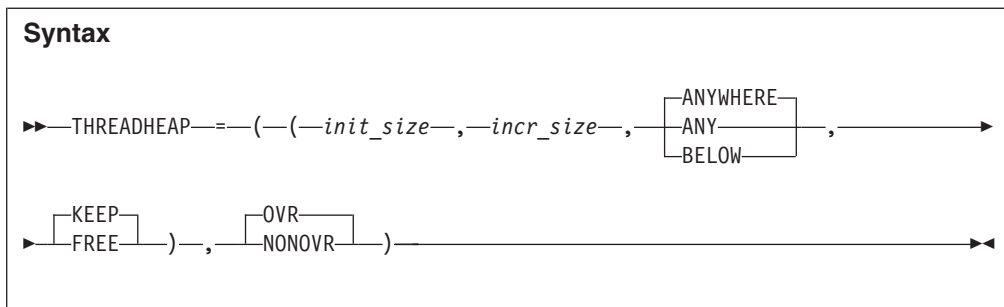
Non-CICS default

```
THREADHEAP=((4K,4K,ANYWHERE,KEEP),OVR)
```

CICS default

THREADHEAP is ignored under CICS.

THREADHEAP



init_size

The minimum initial size of thread heap storage, and is specified in *n*, *nK*, or *nM*. Storage is acquired in multiples of 8 bytes.

A value of zero (0) causes an allocation of 4K.

incr_size

The minimum size of any subsequent increment to the noninitial heap storage is specified in *n*, *nK*, or *nM*. The actual amount of allocated storage is the larger of two values, *incr_size* or the requested size, rounded up to the nearest multiple of 8 bytes.

If you specify *incr_size* as 0, only the amount of the storage needed at the time of the request (rounded up to the nearest 8 bytes) is obtained.

ANYWHERE | ANY

Specifies that the heap storage can be allocated anywhere in storage. If there is no available storage above the line, storage is acquired below the 16-MB line.

The only valid abbreviation of ANYWHERE is ANY.

BELOW

Specifies that the heap storage must be allocated below the 16-MB line.

KEEP

Specifies that storage allocated to THREADHEAP increments is not released when the last of the storage in the thread heap increment is freed.

FREE

Specifies that storage allocated to THREADHEAP increments is released when the last of the storage in the thread heap increment is freed.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

- Even though this option is ignored under CICS, the default increment size under CICS has changed from 4 KB (4096 bytes) to 4080 bytes, to accommodate the 16 byte CICS storage check zone.

Usage notes

- If the requesting routine is running in 24-bit addressing mode and THREADHEAP(,,ANY,) is in effect, THREADHEAP storage is allocated below the 16-MB line based upon the HEAP(,,,,initsz24,incrsz24) settings.

- PL/I MTF considerations — The thread-level heap is allocated only in applications that use the PL/I MTF. For PL/I MTF applications, controlled and based variables specified in subtasks are located in the thread-level heap. If the main program is AMODE 24 and THREADHEAP(,,ANY,) is in effect, heap storage is allocated below the 16-MB line. The only case in which storage is allocated above the line is when all of the following conditions exist:
 - The user routine requesting the storage is running in 31-bit addressing mode.
 - HEAP(,,ANY,,,) is in effect.
 - The main routine is AMODE 31.
- When running PL/I with POSIX(ON) in effect, THREADHEAP is used for allocating heap storage for PL/I base variables declared in non-IPTs. Storage allocated to all THREADHEAP segments is freed when the thread terminates.
- THREADHEAP(4K,4K,ANYWHERE,KEEP) provides behavior compatible with the PL/I TASKHEAP option.
- The initial thread heap segment is never released until the thread terminates.
- THREADHEAP has no effect on C/C++ or VS Fortran MTF applications.

THREADSTACK

Derivation: THREAD level STACK storage

THREADSTACK controls the allocation of the thread's stack storage for both the upward and downward-growing stacks, except the initial thread in a multithreaded application.

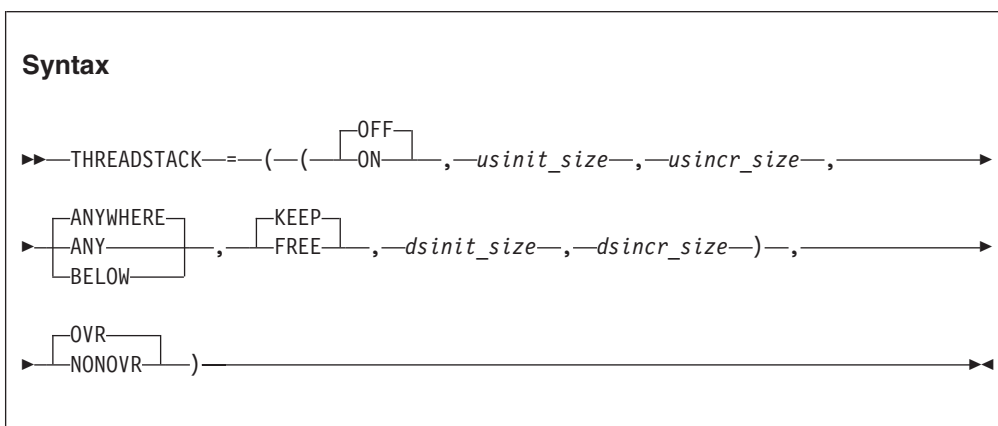
If the thread attribute object does not provide an explicit stack size, then the allocation values can be inherited from the STACK option or specified explicitly on the THREADSTACK option.

Non-CICS default

THREADSTACK=((OFF,4K,4K,ANYWHERE,KEEP,128K,128K),OVR)

CICS default

THREADSTACK is ignored under CICS.



OFF

Indicates that the allocation suboptions of the STACK runtime option are used for thread stack allocation. Any other suboption specified with THREADSTACK is ignored.

THREADSTACK

ON Controls the stack allocation for each thread, except the initial thread, in a multithread environment.

usinit_size

Determines the size of the initial upward-growing stack segment. The storage is contiguous. You specify the *usinit_size* value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

usinit_size can be preceded by a minus sign. In environments other than, if you specify a negative number Language Environment uses all available storage minus the amount specified for the initial stack segment.

A size of "0" or "-0" requests half of the largest block of contiguous storage in the region below the 16-MB line.

usincr_size

Determines the minimum size of any subsequent increment to the upward-growing stack area. You can specify this value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is the larger of two values—*usincr_size* or the requested size—rounded up to the nearest multiple of 8 bytes

If you specify *usincr_size* as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 8 bytes, is obtained.

The requested size is the amount of storage a routine needs for a stack frame. For example, if the requested size is 9000 bytes, *usincr_size* is specified as 8K, and the initial stack segment is full, Language Environment gets a 9000 byte stack increment from the operating system to satisfy the request. If the requested size is smaller than 8K, Language Environment gets an 8K stack increment from the operating system.

ANYWHERE | ANY | BELOW

Specifies the storage location. For downward growing stack, this option is ignored and the storage is always placed above 16MB.

BELOW

Specifies that the stack storage must be allocated below the 16MB line in storage that is accessible to 24-bit addressing.

ANYWHERE | ANY

Specifies that stack storage can be allocated anywhere in storage. If there is no storage available above the line, Language Environment acquires storage below the 16-MB line.

KEEP | FREE

Determines the disposition of the storage increments when the last stack frame in the increment segment is freed.

KEEP

Specifies that storage allocated to stack increments is not released when the last of the storage in the stack increment is freed.

FREE

Specifies that storage allocated to stack increments is released when the last of the storage in the stack is freed. The initial stack segment is never released until the enclave terminates.

dsinit_size

Determines the size of the initial downward growing stack segment. The

storage is contiguous. You specify the *init_size* value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 16 bytes.

dsincr_size

Determines the minimum size of any subsequent increment to the downward growing stack area. You can specify this value as *n*, *nK*, or *nM* bytes of storage. The actual amount of allocated storage is the larger of two values-- *incr_size* or the requested size--rounded up to the nearest multiple of 16 bytes.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- The *dsinit_size* and *dsincr_size* values are the amounts of storage that can be used for downward growing stack frames (plus the stack header, approximately 20 bytes). The actual size of the storage getmained will be 4K (8K if a 4K page alignment cannot be guaranteed) larger to accommodate the guard area.
- The downward growing stack is only initialized in an XPLINK supported environment, and only when an XPLINK application is active in the enclave. Otherwise the suboptions for the downward growing stack are ignored.
- The THREADSTACK option replaces the NONIPTSTACK and NONONIPTSTACK options.
- All storage allocated to THREADSTACK segments are freed when the thread terminates.
- The initial stack segment of the thread is never released until the thread terminates, regardless of the KEEP/FREE state.
- You can specify suboptions with THREADSTACK(OFF,...), but they are ignored. If you override the THREADSTACK(OFF,...) suboption with THREADSTACK(ON) and you omit suboptions, then the suboptions you specified with THREADSTACK(OFF,...) remain in effect. If you respecify THREADSTACK(OFF,...) with different suboptions, they override the defaults.
- In the multithreaded environment, you can explicitly specify the stack size in the thread attribute object; it will be used instead of the value specified with THREADSTACK or STACK.

PL/I MTF considerations

THREADSTACK(ON,4K,4K,BELOW,KEEP,,) provides PL/I compatibility for stack storage allocation and management for each subtask in the application.

PL/I considerations

For multitasking or multithreaded environments, the stack size for a subtask or non-Initial Process Thread (non-IPT) is taken from the THREADSTACK option unless THREADSTACK(OFF) is specified. THREADSTACK(OFF) specifies that the values in the STACK option be used.

For more information

- For more information about the STACK runtime option, see “STACK” on page 113.
- For more information about the ALL31 runtime option, see “ALL31” on page 50.

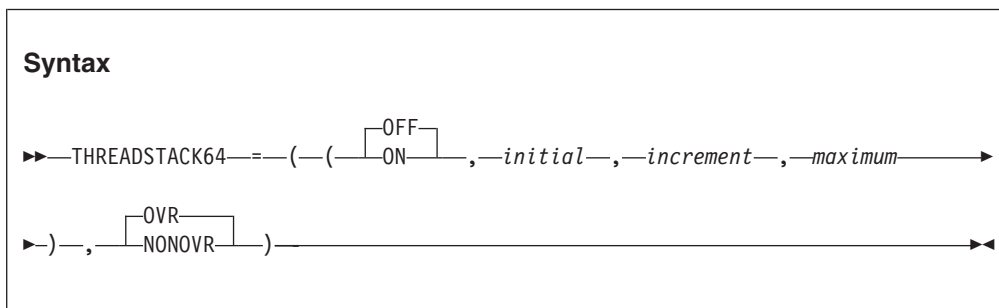
THREADSTACK64 (AMODE 64 only)

Derivation: THREAD level STACK storage for AMODE 64

THREADSTACK64 controls the allocation of the thread's stack storage for AMODE 64 applications, except for the initial thread in a multithreaded environment.

AMODE 64 default

THREADSTACK64=((OFF,1M,1M,128M),OVR)



OFF

Indicates that the allocation suboptions of the STACK64 runtime option are used for thread stack allocation. Any other suboption specified with THREADSTACK64 is ignored.

ON Controls the stack allocation for each thread, except the initial thread, in a multithreaded environment.

initial

Determines the size of the initial stack segment. The storage is contiguous. This value is specified as *nM* bytes of storage.

increment

Determines the minimum size of any subsequent increment to the stack area. This value is specified as *nM* bytes of storage. The actual amount of allocated storage is the larger of two values— *increment* or the requested size—rounded up to the nearest multiple of 1MB.

If you specify *increment* as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 1MB, is obtained.

The requested size is the amount of storage a routine needs for a stack frame.

maximum

Specifies the maximum stack size. This value is specified as *nM* bytes of storage. When the maximum size is less than the initial size, *initial* is used as the maximum stack size.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- The 1MB guard area is not included in any of the sizes.

- The maximum thread stack segment is the maximum of THREADSTACK64 initial and maximum sizes.
- When a multithreaded application that creates many pthreads is run, the default value of 128MB for the maximum stack size of the STACK64 and THREADSTACK64 runtime options might cause excessive use of system resources, such as real storage. For such applications, you need to use the Language Environment Storage Report (RPTSTG runtime option) to determine the actual pthread stack storage usage of your application, and then use the THREADSTACK64 runtime option to set the maximum stack size to a value closer to the actual usage.

Performance considerations

To improve performance, use the storage report numbers generated by the RPTSTG runtime option as an aid in setting the initial and increment sizes for THREADSTACK64.

For more information

- See “RPTSTG” on page 108 for more information about the RPTSTG runtime option.
- For more information about using the storage reports generated by the RPTSTG runtime option to tune the stacks for AMODE 64 applications, see *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*.

TRACE

TRACE controls runtime library tracing activity, the size of the in-storage trace table, the type of trace events to record, and it determines whether a dump containing, at a minimum, the trace table should be unconditionally taken when the application terminates. When you specify TRACE(ON), user-requested trace entries are intermixed with Language Environment trace entries in the trace table.

Under normal termination conditions, if TRACE is active and you specify DUMP, only the trace table is written to the dump report, independent of the TERMTHDACT setting. Only one dump is taken for each termination. Under abnormal termination conditions, the type of dump taken (if one is taken) depends on the value of the TERMTHDACT runtime option and whether TRACE is active and the DUMP suboption is specified.

Non-CICS default

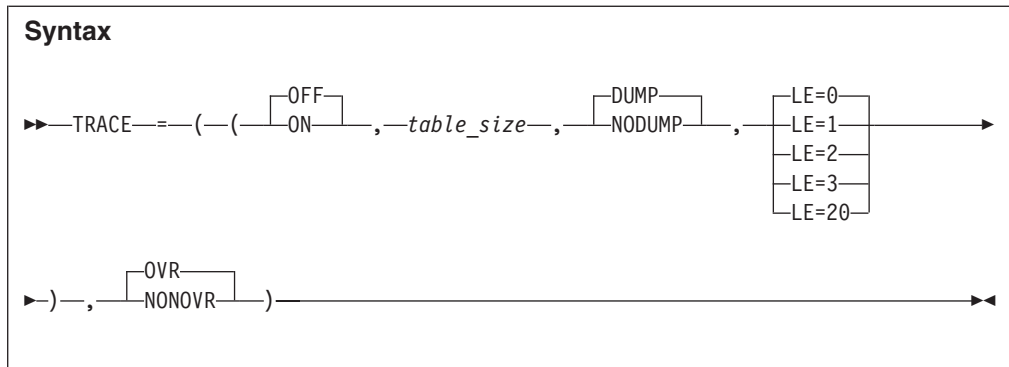
TRACE=((OFF,4K,DUMP,LE=0),OVR)

CICS default

TRACE=(OFF,4K,DUMP,LE=0)

AMODE 64 default

TRACE=(OFF,,DUMP,LE=0)

**OFF**

Indicates that the tracing facility is inactive.

ON Indicates that the tracing facility is active.

table_size

Determines the size of the tracing table as specified in bytes (*nK* or *nM*). The upper limit is 16M - 1 (16777215 bytes).

Restriction: This suboption is ignored for AMODE 64 applications and the size is set to 1M.

DUMP

Requests that a Language Environment-formatted dump (containing the trace table) be taken at program termination regardless of the setting of the TERMTHDACT runtime option.

NODUMP

Requests that a Language Environment-formatted dump not be taken at program termination.

LE=0

Specifies that no trace events be recorded.

LE=1

Specifies that entry to and exit from Language Environment member libraries be recorded (such as, in the case of C, entry and exit of the printf() library function).

LE=2

Specifies that mutex init/destroy and locks/unlocks from Language Environment member libraries be recorded.

LE=3

Activates both the entry/exit trace and the mutex trace.

LE=20

Specifies that XPLINK/non-XPLINK transition should be recorded.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

Usage notes

- When running PL/I with POSIX(ON), no PL/I-specific trace information is provided.
- When you specify LE=20:

- AMODE 64 applications have no transitions.
- Transitions across OS_UPSTACK linkage are not recorded.
- Under abnormal termination, the following dump contents are generated:

TERMTHDACT(TRACE)

Generates a dump containing the trace table and the traceback and options report.

TERMTHDACT(QUIET)

Generates a dump containing the trace table only.

TERMTHDACT(MSG)

Generates a dump containing the trace table only.

TERMTHDACT(DUMP)

Generates a dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block) and an options report.

TERMTHDACT(UADUMP)

Generates a system dump of the user address space and an options report.

PL/I MTF considerations

The TRACE(ON,,,LE=2) setting provides the following trace table entries for PL/I MTF support:

- Trace entry 100 occurs when a task is created.
- Trace entry 101 occurs when a task that contains the tasking CALL statements is terminated.
- Trace entry 102 occurs when a task that does not contain the tasking CALL statements is terminated.

For more information

- For more information about the dump contents, see “TERMTHDACT” on page 121.
- For more information about using the tracing facility, see *z/OS Language Environment Debugging Guide*.

TRAP

TRAP specifies how Language Environment programs handle abends and program interrupts.

TRAP(ON) must be in effect for the ABTERMENC run-time option to have effect.

This option is similar to the STAE | NOSTAE run-time option currently offered by COBOL, C, and PL/I, and the SPIE | NOSPIE option offered by C and PL/I:

Table 11. TRAP runtime option settings

If	Then
A single option is specified in input,	TRAP is set according to that option, TRAP(OFF) for NOSTAE or NOSPIE, TRAP(ON) for STAE or SPIE.
Both options are specified in input,	TRAP is set ON, unless both options are negative. TRAP is set OFF if both options are negative.

TRAP

Table 11. TRAP runtime option settings (continued)

If	Then
STAE is specified in one #pragma runopts statement, and NOSPIE in another,	The option in the last #pragma runopts determines the setting of TRAP.
Multiple instances of STAE NOSTAE are specified,	TRAP is set according to the last instance only. All others are ignored.
Multiple instances of SPIE NOSPIE are specified,	TRAP is set according to the last instance only. All others are ignored.
An options string has TRAP(ON) or TRAP(OFF) together with SPIE NOSPIE, and/or STAE NOSTAE,	The TRAP setting takes preference over all others.

CEESGL is unaffected by this option.

Non-CICS default

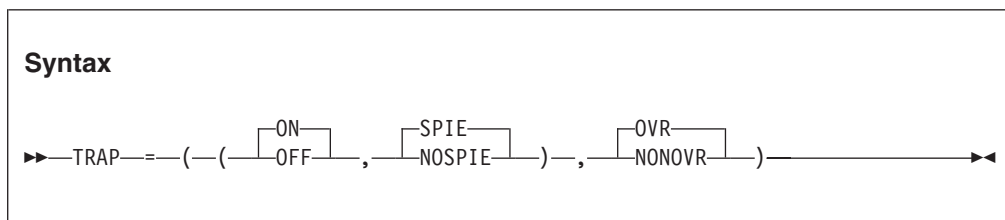
TRAP=((ON,SPIE),OVR)

CICS default

TRAP=((ON,SPIE),OVR)

Amode 64 default

TRAP=((ON,SPIE),OVR)



ON Fully enables the Language Environment condition handler.

OFF

Prevents language condition handlers or handlers registered by CEEHDLR from being notified of abends or program checks; prevents application of POSIX signal handling semantics for abends and program checks.

SPIE

SPIE specifies that Language Environment issue an ESPIE macro to handle program interrupts. The SPIE suboption is ignore when specified with the OFF suboption.

NOSPIE

NOSPIE specifies that Language Environment will not issue the ESPIE macro. When you specify the ON suboption, Language Environment handles program interrupts and abends using an ESTAE. The NOSPIE suboption is ignored when specified with the OFF suboption.

Due to the restrictions and side effects when running TRAP(OFF) stated in "Usage notes" on page 139, IBM highly recommends running TRAP(ON,SPIE) in all environments.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

Because Language Environment never sets a SPIE or STAE, the SPIE | NOSPIE suboption is ignored on CICS.

z/OS UNIX considerations

The TRAP option applies to the entire enclave and all threads within.

Usage notes

- **Restriction:** TRAP(OFF) is not supported for AMODE 64 applications.
 - The SPIE | NOSPIE runtime option offered by C and PL/I does not affect the TRAP suboptions SPIE and NOSPIE.
 - Use TRAP(OFF) only when you need to analyze a program exception before Language Environment handles it.
 - When you specify TRAP(OFF) in a non-CICS environment, an ESPIE is not issued, but an ESTAE is issued. Language Environment does not handle conditions raised by program interrupts or abends initiated by SVC 13 as Language Environment conditions, and does not print messages for such conditions.
 - Running with TRAP(OFF) (for exception diagnosis purposes) can cause many side effects, because Language Environment uses condition handling internally and requires TRAP(ON). When you run with TRAP(OFF), you can get side effects even if you do not encounter a software-raised condition, program check, or abend. If you do encounter a program check or an abend with TRAP(OFF) in effect, the following side effects can occur:
 - Fixed-point overflow exceptions are not ignored when the PSW mask is ON. C/C++ and COBOL language semantics expect the exceptions to be ignored. When PL/I is part of the application, the PSW mask is ON.
 - The ABTERMENC runtime option has no effect.
 - The ABPERC runtime option has no effect.
 - Resources acquired by Language Environment are not freed.
 - Files opened by HLLs are not closed by Language Environment, so records might be lost.
 - The abnormal termination exit is not driven for enclave termination.
 - The assembler user exit is not driven for enclave termination.
 - User condition handlers are not enabled.
 - The debugger is not notified of the error.
 - No storage report or runtime options report is generated.
 - No Language Environment messages or Language Environment dump output is generated.
 - In z/OS UNIX, POSIX signal handling semantics are not enabled for the abend.
- The enclave terminates abnormally if such conditions are raised.
- TRAP(ON) must be in effect when you use the CEEBXITA assembler user exit for enclave initialization to specify a list of abend codes that Language Environment percolates.

TRAP

- When TRAP(ON) is in effect, and the abend code is in the CEEAUE_A_AB_CODES list in CEEBXITA, Language Environment percolates the abend. Normal Language Environment condition handling is never invoked to handle these abends. This feature is useful when you do not want Language Environment condition handling to intervene for certain abends or when you want to prevent invocation of the abnormal termination exit for certain abends, such as when IMS issues a user ABEND code 777.
- When TRAP(ON,NOSPIE) is specified, Language Environment will handle program interrupts and abends via an ESTAE. This feature is useful when you do not want Language Environment to issue an ESPIE macro. If you do not want Language Environment to issue an ESPIE, you must specify TRAP(OFF). When TRAP(OFF), (TRAP(OFF,SPIE) or TRAP(OFF,NOSPIE) is specified and there is a program interrupt, the user exit for termination is not driven.

C++ considerations

TRAP(ON) must be in effect in order for the z/OS C++ try/throw/catch condition handling mechanisms to work.

For more information

- See “ABTERMENC” on page 47 for more information about the ABTERMENC runtime option.
- See *z/OS Language Environment Programming Reference* for more information about the CEESGL callable service, or the CEEHDLR callable service.
- See *z/OS Language Environment Programming Guide* for more information about the CEEBXITA assembler user exit.

UPSI (COBOL only)

Derivation: User Programmable Status Indicator

UPSI sets the eight UPSI switches on or off for applications that use COBOL programs.

Non-CICS default

UPSI=((00000000),OVR)

CICS default

UPSI=((00000000),OVR)

Syntax

```
>>—UPSI—=—(—(—nnnnnnnn—), —[OVR / NONOVR]—)——>>
```

nnnnnnnn

n represents one UPSI switch between 0 and 7, the leftmost *n* representing the first switch. Each *n* can either be 0 (off) or 1 (on).

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

For more information

- For more information about how COBOL programs access the UPSI switches, see the appropriate version of the programming guide in the COBOL library at Enterprise COBOL for z/OS library (<http://www-01.ibm.com/support/docview.wss?uid=swg27036733>).

USRHDLR

Derivation: USeR condition HanDLeR

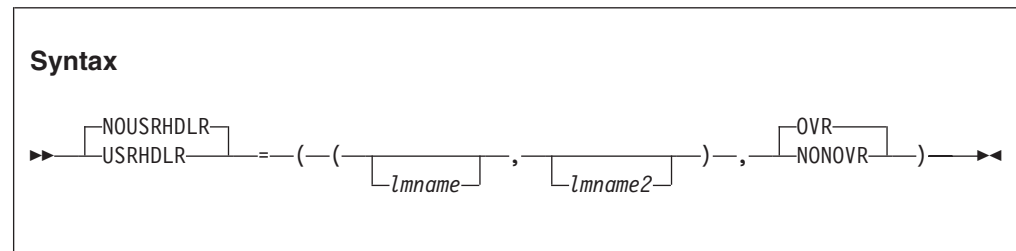
USRHDLR registers a user condition handler at stack frame 0, allowing you to register a user condition handler without having to include a call to CEEHDLR in your application and then recompile the application.

Non-CICS default

NOUSRHDLR=((),OVR)

CICS default

NOUSRHDLR=((),OVR)



NOUSRHDLR

Does not register a user condition handler without recompiling an application to include a call to CEEHDLR.

USRHDLR

Registers a user condition handler without recompiling an application to include a call to CEEHDLR.

lmtime

The name of a load module (or an alias name of a load module) that contains the user condition handler that is to be registered at stack frame 0. This parameter is optional.

lmtime2

The name of a load module (or an alias name of a load module) that contains the user condition handler that is to be registered to get control after the enablement phase and before any other user condition handler. This parameter is optional.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

CICS considerations

When specifying USRHDLR under CICS, *lmtime* and *lmtime2* must be defined in the CICS PPT.

Usage notes

- The user condition handler specified by the USRHDLR runtime option must be in a separate load module rather than be link-edited with the rest of the application.
- The user condition handler *lmname* is invoked for conditions that are still unhandled after being presented to condition handlers for the main program.
- The user condition handler *lmname2* is invoked for each condition after the condition completes the enablement phase but before any other registered user condition handlers are given control.
- You can use a user condition handler registered with the USRHDLR runtime option to return any of the result codes allowed for a user condition handler registered with the CEEHDLR callable service.
- A condition that is percolated or promoted by a user condition handler registered to handle conditions at stack frame 0 using the USRHDLR runtime option is not presented to any other user condition handler.
- The loading of the user condition handlers *lmname* and *lmname2* occurs only when that user condition handler needs to be invoked the first time.
- If the load of either *lmname* or *lmname2* fails, an error message is issued.
- To turn off one of the suboptions previously specified by USRHDLR (*lmname* or *lmname2*), specify the option with either empty single quotation marks or empty double quotation marks. For example, to turn off the *lmname2* suboption after it had been previously specified, use either `USRHDLR(lmname, '')` or `USRHDLR(lmname, "")`.
- IBM supplies a sample user-written condition handler found in SCEESAMP called CEEWUCHA. Under CICS, this handler will give you similar abend codes that were around in certain pre-Language Environment environments. The CEEWUCHA load module needs to be built using CEEWWCHA provided in SCEESAMP. Be aware that this handler has support for both COBOL and PL/I and is shipped with the PL/I specific behavior commented out. If you want this PL/I behavior, modify the source before using CEEWWCHA.

For more information

For information about registering a user condition handler and its interfaces, see the CEEHDLR callable service in *z/OS Language Environment Programming Reference*.

VCTRSAVE

Derivation: VeCToR environment to be SAVEd

VCTRSAVE specifies whether any language in the application uses the vector facility when user-written condition handlers are called.

Non-CICS default

VCTRSAVE=((OFF),OVR)

CICS default

VCTRSAVE is ignored under CICS.

Syntax

```

▶▶ VCTRSAVE = ( ( OFF | ON ) , ( OVR | NONOVR ) ) ▶▶

```

OFF

No language in the application uses the vector facility when user-provided condition handlers are called.

ON

A language in the application uses the vector facility when user-provided condition handlers are called.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

The VCTRSAVE option applies to the entire enclave and all threads within.

Performance considerations

When a condition handler plans to use the vector facility (that is, run any vector instructions), the entire vector environment has to be saved on every condition and restored on return to the application code. You can avoid this extra work by specifying VCTRSAVE(OFF) when you are not running an application under vector hardware.

XUFLOW**Derivation:** eXponent Under FLOW

XUFLOW specifies whether an exponent underflow causes a program interrupt. An exponent underflow occurs when a floating point number becomes too small to be represented.

The underflow setting is determined at enclave initialization and is updated when new languages are introduced into the application (via fetch or dynamic call, for example). Otherwise, it does not vary while the application is running.

Language Environment preserves the language semantics for C/C++ and COBOL regardless of the XUFLOW setting. Language Environment preserves the language semantics for PL/I only when XUFLOW is set to AUTO or ON. Language Environment does not preserve the language semantics for PL/I when XUFLOW is set to OFF.

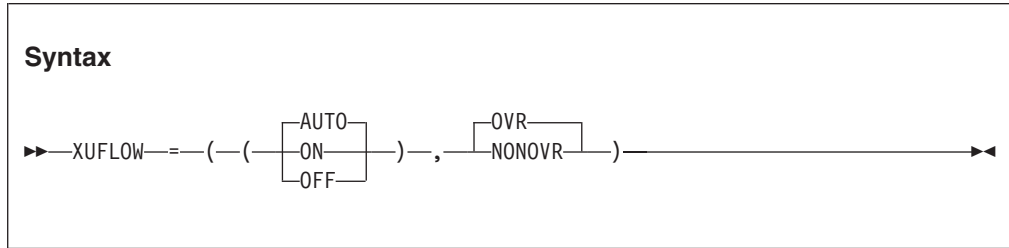
An exponent underflow caused by a C/C++ or COBOL program does not cause a condition to be raised.

Non-CICS default

XUFLOW=((AUTO),OVR)

CICS default

XUFLOW=((AUTO),OVR)



AUTO

An exponent underflow causes or does not cause a program interrupt dynamically, based upon the HLLs that make up the application. Enablement is determined without user intervention.

XUFLOW(AUTO) causes condition management to process underflows only in those applications where the semantics of the application languages require it. Normally, XUFLOW(AUTO) provides the best efficiency while meeting language semantics.

ON An exponent underflow causes a program interrupt.

XUFLOW(ON) causes condition management to process underflows regardless of the mix of languages; therefore, this setting might be less efficient in applications that consist of languages not requiring underflows to be processed by condition management.

OFF

An exponent underflow does not cause a program interrupt; the hardware takes care of the underflow.

When you set XUFLOW to OFF, the hardware processes exponent underflows. This is more efficient than condition handling to process the underflow.

OVR

Specifies that the option can be overridden.

NONOVR

Specifies that the option cannot be overridden.

z/OS UNIX considerations

The XUFLOW option applies to the entire enclave and all threads within.

Usage notes

PL/I considerations

When setting XUFLOW to OFF, be aware that the semantics of PL/I require the underflow to be signaled.

Chapter 7. Customizing user exits

Language Environment provides support for the following user exits:

Assembler user exit

Performs functions for enclave initialization, normal and abnormal enclave termination, and process termination. See “Changing the assembler language user exit” on page 146.

High-level language (HLL) user exit

Performs functions for enclave initialization. See “Changing the high-level language user exit” on page 148.

Abnormal termination user exit

Collects problem determination data when Language Environment is terminating an enclave due to an unhandled condition. See “Customizing Language Environment abnormal termination exits” on page 149.

Load notification user exit

Improves performance by preventing frequently used modules from being loaded and deleted with each use. See “Creating a load notification user exit” on page 154. The load notification user exit is only available when Library Routine Retention (LRR) is used.

Storage tuning user exit

Provides a programming interface for collecting Language Environment storage tuning information and setting the Language Environment runtime option values for STACK, LIBSTACK, HEAP, ANYHEAP and BELOWHEAP. See “Creating a storage tuning user exit” on page 156. The storage tuning user exit is available for CICS, and for non-CICS environments when LRR is used.

Restriction: Only the abnormal termination user exit supports AMODE 64 applications.

See “Storage tuning user exit” on page 198 for more information about the features of the exits, default values, and syntax.

Choose which sample customization jobs to modify and run. Table 12 lists the sample jobs that are members of Language Environment sample library SCEESAMP.

Table 12. Sample customization jobs for the user exits

Use this sample job	To
CEEWDXIT	Change installation-wide assembler language user exit.
CEEWXCIT	Change installation-wide CICS assembler language user exit.
CEEWUXIT	Create an application-specific assembler language user exit.
CEEWHLX	Change high-level language user exit.
CEEWDEXT	Identify an abnormal termination exit (non-CICS).
CEEWCEXT	Identify an abnormal termination exit (CICS).
CEEWQEXT	Identify an abnormal termination exit (AMODE 64).
CEEWLNUE	Identify a load notification user exit.

Example

If there is an unhandled condition of severity 2 or greater, the default assembler user exit in z/OS returns to the system with a return code. You can change the default assembler user exit so that it forces an abend for unhandled conditions of severity 2 or greater.

Examples of conditions that are severity 2 or greater include:

- Program interrupts
- System abends
- Conditions detected by Language Environment; for example, a program load failure

The ABTERMENC(ABEND) runtime option is an alternative way to force an abend for unhandled conditions of severity 2 or greater.

Changing the assembler language user exit

Three sample jobs are installed in the CEE.SCEESAMP target data set to help you modify the assembler language user exit. Two of the jobs use SMP/E USERMODs to replace the IBM-supplied installation-wide assembler user exits. The third sample job creates an application-specific assembler user exit that can be link-edited with applications that need its functions. You can create several different application-specific user exits, each in a different partitioned data set, to satisfy the needs of different application programs. Source code for the sample assembler user exits is installed as members in the CEE.SCEESAMP data set.

Table 13. Sample assembler user exits for Language Environment

Example user exit	Operating system	Language (if language-specific)
CEEBXITA	z/OS (default)	
CEEBXITC	TSO/E	
CEECXITA	CICS (default)	
CEEBX05A	z/OS	VS COBOL II compatibility

Note:

1. CEEBXITA and CEECXITA are the defaults on your system for z/OS and CICS, if Language Environment is installed at your installation without modification.
2. The source code for CEEBXITA, CEEBXITC, CEECXITA, and CEEBX05A can be found in the SCEESAMP sample library.

Use the information in *z/OS Language Environment Programming Guide* to assist you in modifying the IBM-supplied user exits or in creating your own.

If you specify runtime options in an assembler language user exit, they override all other sources of runtime options except those that are specified as NONOVR.

CEEBXITA performs functions for enclave initialization, normal and abnormal enclave termination, and process termination. CEEBXITA must be written in assembler language, because an HLL environment might not be established when the exit is invoked.

You can set up user exits for tasks such as:

- Installation accounting and charge back
- Installation audit controls

- Programming standard enforcement
- Common application runtime support

Changing the installation-wide assembler language user exit (non-CICS)

Use the CEEWDXIT sample job to change the installation-wide assembler language user exit. You must replace the comment in CEEWDXIT with your source for CEEBXITA. You can copy the source for the IBM-supplied default installation-wide assembler language user exit from CEEBXITA in CEE.SCEESAMP and modify it to suit your needs, or you can create your own source for CEEBXITA. Use the information in *z/OS Language Environment Programming Guide* to guide you in coding your changes.

To modify the JCL for CEEWDXIT:

1. Replace the comment lines following the ++ SRC statement in the job with your source program for the installation-wide assembler language user exit.
2. Change #GLOBALCSI to the data set name of your global CSI data set.
3. Change #TZONE to the name of your target zone.
4. Add necessary SMP/E PRE information for PTFs you have installed on your system which contain the same part.

CEEWDXIT should run with a condition code of 0.

Changing the installation-wide assembler language user exit (CICS)

Use the CEEWCXIT sample job to change the CICS installation-wide assembler language user exit. You must replace the comment in CEEWCXIT with your source for CEECXITA. You can copy the source for the IBM-supplied default installation-wide assembler language user exit from CEECXITA in CEE.SCEESAMP and modify it to suit your needs, or you can create your own source for CEECXITA.

Note the difference between the IBM-supplied CEEBXITA and the IBM-supplied CEECXITA. You can retain some or all of these differences in your user exit. Use the information in *z/OS Language Environment Programming Guide* to guide you in coding your changes.

To modify the JCL for CEEWCXIT:

1. Replace the comment lines following the ++ SRC statement in the job with your source program for the installation-wide CICS assembler language user exit.
2. Change #GLOBALCSI to the data set name of your global CSI data set.
3. Change #TZONE to the name of your target zone.
4. Add necessary SMP/E PRE information for PTFs you have installed on your system which contain the same part.

CEEWCXIT should run with a condition code of 0.

Note: CICS TS 3.1 and higher supports XPLINK programs in a CICS environment. The installation-wide assembler user exit for non-CICS is used for these programs.

Creating an application-specific assembler language user exit

Use the CEEWUXIT sample job to create as many application-specific assembler language user exits as your site requires. You must replace the comment in CEEWUXIT with your source. You can copy the source for the IBM-supplied default installation-wide assembler language user exit from CEEBXITA or CEEBXITC in CEE.SCEESAMP and modify it to suit your needs, or you can create your own source.

CEEWUXIT does not use SMP/E to create the assembler language user exit module, so it can be run several times to create several different CEEBXITA modules, each in its own user-specified library. Use the information in *z/OS Language Environment Programming Guide* to guide you in coding your changes.

Steps for modifying the JCL for CEEWUXIT

Perform the following steps to modify the JCL for CEEWUXIT:

1. Replace the comment lines following the //SYSIN statement in the job with your source program for the application-specific assembler language user exit.

2. Change DSN=YOURLIB in the SYSLMOD DD statement to the name of the partitioned data set you want your CEEBXITA module link-edited into.
Note: A CEEBXITA module currently in the chosen data set is replaced by the new version.

3. Check the SYSLIB DD statement to ensure that the data set names are correct.

4. Bind (link) the resultant CEEBXITA module with your application.

When you are done, CEEWUXIT should run with a condition code of 0.

Exception: If your exit is written in C/C++, you could get a condition code of 4 if your job runs correctly.

Changing the high-level language user exit

The CEEWHLLX sample job contains an SMP/E USERMOD that replaces the IBM-supplied high-level language user exit with your high-level language user exit. The USERMOD contains the object program for the user exit, not the high-level language source.

SMP/E is not able to compile a source language other than assembler language, so you must compile your user exit and place the object program produced by the compiler into the USERMOD in CEEWHLLX. See *z/OS Language Environment Programming Guide* for a description of the high-level language user exit interface.

If you write your high-level language user exit in C/C++, use the #pragma csect statement to name the CSECT CEEBINT. Use the #pragma map statement to instruct the compiler to correctly convert references to CEEBINT as follows:

```
#pragma map(CEEBINT,"CEEBINT")
```

You can also write high-level language user exits in PL/I and Language Environment-conforming assembler.

If you use any of the C/C++ library functions, the CEEWHLLX job might generate the following message.

```
IEW2454W nnnn SYMBOL xxxxxxxx UNRESOLVED.  
NO AUTOCALL (NCAL) SPECIFIED.
```

Although you might receive a condition code of 04, this code does not indicate an error.

Steps for modifying the JCL for CEEWHLLX

Perform the following steps to modify the JCL for CEEWHLLX

1. Replace the comment lines following the ++ MOD statement in CEEWHLLX with the object program obtained by compiling your high-level language user exit.

2. Change #GLOBALCSI to the data set name of your global CSI data set.

3. Change #TZONE to the name of your target zone.

4. Add necessary SMP/E PRE information for PTFs you have installed on your system which contain the same part.

When you are done, CEEWHLLX should run with a condition code of 0.

Exception: If your exit is written in C/C++, you could get a condition code of 4 if your job runs correctly.

Customizing Language Environment abnormal termination exits

If Language Environment encounters an unhandled condition of severity 2 or greater, it can invoke an abnormal termination exit before it terminates the enclave. If the abnormal termination exit is invoked before the thread is terminated, the abnormal termination exit can collect problem determination data before Language Environment frees the resources that it acquired.

To generate a system dump of the user address space, you can use the TERMTHDACT(UADUMP) runtime option.

The CEEEXTAN (non-CICS), CEECXTAN (CICS), and CELQXTAN (AMODE 64) CSECTs, which are installed in the CEE.SCEESAMP target data set, contain the instructions for defining which abnormal termination exits, if any, are called when a routine terminates abnormally. Use the CEEWDEXT (non-CICS), CEEWCEXT (CICS), and CEEWQEXT (AMODE 64) sample jobs to replace the existing CSECTs with your updated CSECTs in your runtime library. For the syntax and other considerations for abnormal termination exits, see *z/OS Language Environment Programming Guide*.

Note: CICS TS 3.1 and higher supports XPLINK programs in a CICS environment. The abnormal termination exit for non-CICS (CEEEXTAN) is used for these programs.

Creating a Language Environment abnormal termination exit

To create an abnormal termination exit:

1. Create an assembler language routine that conforms to the syntax described in *z/OS Language Environment Programming Guide* or *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*. AMODE 64 abnormal termination exit routines should specify the "FETCHABLE=RENT" option on the CELQRLG MACRO. See CEEWQATX in CEE.SCEESAMP for an example to use with AMODE 64 applications.
2. Assemble and link-edit your exit into a library that Language Environment can access at runtime, such as SCEERUN or SCEERUN2.
3. Code a CEEEXTAN CSECT that contains a CEEXART macro identifying your exit. The macro specifies your routine as an abnormal termination exit routine. The CEEEXTAN CSECT can be found in source file CEEEXTAN (for CICS), CEEEXTAN (for non-CICS), or CELQXTAN (for AMODE 64). See "CEEEXTAN abnormal termination exit CSECT" for more information.
4. Replace the existing CEEEXTAN CSECT with the updated CEEEXTAN as described in the following sections.

CEEEXTAN abnormal termination exit CSECT

CEEEXTAN is a CSECT explicitly linked with the Language Environment condition handling routines, and it is the CSECT that you create by coding the CEEXAHD, CEEXART, and CEEXAST macros. Specifically, CEEEXTAN is linked with the CEEPLPKA, CEECCICS, and CELQLIB load modules. CEEEXTAN CSECT is created through the use of the following Language Environment-provided assembler macros:

CEEXAHD

Defines the header of the table. CEEXAHD generates the CSECT statement and any header information required. CEEXAHD uses an amode operand, which can be specified as AM=ANY or AM=64.

CEEXART

Identifies the name of the abnormal termination exit to be invoked. It generates one entry for an abnormal termination exit. It has only one keyword parameter, TERMXIT=, which is the load name for the abnormal termination exit. There is a limit of 8 characters for the load name, and no validation of the name is performed by the macro.

More than one invocation of CEEXART can appear in the CEEEXTAN CSECT, thus allowing multiple abnormal termination exits to be registered. When more than one name is specified, the abnormal termination exits are honored in the order found in the CEEEXTAN CSECT.

CEEXAST

Identifies the end of the list of abnormal termination exits. It generates the trailer for the CEEEXTAN CSECT. It has no parameters.

Language Environment validates the format of the abnormal termination exit CSECT and issues a load of the names as identified in the table. The LOAD is attempted only for terminations due to unhandled conditions of severity 2 or greater. If the LOAD is successful, an abnormal termination exit is invoked according to the interface described in the following sections. If the LOAD fails (the routine cannot be found, or there is not enough storage for the routine, for example), no error indication is delivered and either the next name in CEEEXTAN is chosen, or termination continues (if the names were exhausted). This allows a STEPLIB to either contain or omit the load names, depending on whether you want the exit to be used for this job.

Jobs to generate and modify CEEEXTAN CSECT

You can use three source files to generate CEEEXTAN CSECT, one for CICS, one for non-CICS, and one for AMODE 64 applications. The following source files are provided in the SCEESAMP data set:

CEEEXTAN

Source to generate CEEEXTAN CSECT for CICS

CEEEXTAN

Source to generate CEEEXTAN CSECT for non-CICS

CELQXTAN

Source to generate CEEEXTAN CSECT for AMODE 64

You can use the following two jobs to replace CEEEXTAN CSECT:

CEEWCEXT

Replaces CEEEXTAN CSECT for CICS

CEEWDEXT

Replaces CEEEXTAN CSECT for non-CICS

CEEWQEXT

Replaces CEEEXTAN CSECT for AMODE 64

Figure 2 contains the source for the IBM-supplied CEEEXTAN:

```
                TITLE 'LE/370 Abnormal Termination User exit CSECT'
                CEEAHD      ,User exit header
*
*****
* To specify an abnormal termination exit, change the line
* where CEEART is specified:
* - change the XXXXXXXX to the name of the abnormal termination exit
* - change the '*' in column 1 to a blank
*****
*      CEEART  TERMxit=XXXXXXX
*
                CEEAST      ,Terminate the list
```

Figure 2. Default CEEEXTAN

If you want to add your own abnormal termination exit called WHODIDIT, then the code should look like the following example:

```
                TITLE 'LE/370 Abnormal Termination User exit CSECT'
                CEEAHD      ,User exit header
*
*****
* To specify an abnormal termination exit, change the line
* where CEEART is specified:
* - change the XXXXXXXX to the name of the abnormal termination exit
* - change the '*' in column 1 to a blank
*****
                CEEART  TERMxit=WHODIDIT
*
                CEEAST      ,Terminate the list
```

Figure 3. Updated CEEEXTAN

Identifying the abnormal termination exit (non-CICS)

Use the CEEWDEXT sample job to specify your own abnormal termination exit in a non-CICS environment.

Steps for modifying the JCL for CEEWDEXT

Perform the following steps to modify the JCL for CEEWDEXT:

1. Replace the comment lines following the ++ SRC statement in CEEWDEXT with your updated CEEEXTAN CSECT identifying your abnormal termination exit routine.

2. Change #GLOBALCSI to the data set name of your global CSI data set.

3. Change #TZONE to the name of your target zone.

4. Add necessary SMP/E PRE information for PTFs you have installed on your system which contain the same part.

When you are done, CEEWDEXT should run with a condition code of 0.

Identifying the abnormal termination exit (CICS)

Use the CEEWCEXT sample job to specify your own abnormal termination exit in a CICS environment.

Steps for modifying the JCL for CEEWCEXT

Perform the following steps to modify the JCL for CEEWCEXT:

1. Replace the comment lines following the ++ SRC statement in CEEWCEXT with your updated CEEEXTAN CSECT identifying your abnormal termination exit routine.

2. Change #GLOBALCSI to the data set name of your global CSI data set.

3. Change #TZONE to the name of your target zone.

4. Add necessary SMP/E PRE information for PTFs you have installed on your system which contain the same part.

When you are done, CEEWCEXT should run with a condition code of 0.

Identifying the abnormal termination exit (AMODE 64)

Use the CEEWQEXT sample job to specify your own abnormal termination exit for AMODE 64 applications.

Steps for modifying the JCL for CEEWQEXT

Perform the following steps to modify the JCL for CEEWQEXT:

1. Replace the comment lines following the ++ SRC statement in CEEWQEXT with your updated CELQXTAN identifying your abnormal termination exit routine.

2. Change #GLOBALCSI to the data set name of your global CSI data set.

3. Change #TZONE to the name of your target zone.

-
4. Add necessary SMP/E PRE information for PTFs you have installed on your system which contain the same part.
-

When you are done, CEEWQEXT should run with a condition code of 0.

Creating global user exit XPCFTCH (CICS)

The CICS global user exit XPCFTCH allows an assembler user exit to put its own entry point in place of the current known entry for a routine by returning a new address in PCUE_BRANCH_ADDRESS. The CICS XPCFTCH exit enhancement for z/OS V1R7 requires CICS Transaction Server for z/OS Version 2.3 with PTF UQ95648.

Using XPCFTCH for an Enterprise PL/I routine

When the CICS global user exit XPCFTCH is used to intercept an Enterprise PL/I routine by providing an alternate entry point, CICS uses the address returned in `ceecics_rumentry_real` if it is available. It is supplied to XPCFTCH as the new PCUE_REAL_ENTRY parameter. You can also provide a MAIN that conforms to Language Environment with the hexadecimal value `x'47F0F014'` followed by `x'01'CEE`, DSASIZE, and the offset to the PPA1. Refer to the common interfaces and conventions information in *z/OS Language Environment Vendor Interfaces* for details about routine layout.

Using XPCFTCH for a PL/I routine

When the CICS global user exit XPCFTCH is used to intercept a PL/I routine by providing an alternate entry point, CICS uses the address returned in `ceecics_rumentry_real` if it is available. It is supplied to XPCFTCH as the new PCUE_REAL_ENTRY parameter. You can also provide a Language Environment CEESTART with the following:

- The CEESTART eyecatcher
- A pointer to a CEEMAIN with a MAIN address pointing to your replaced MAIN

There is no need for a CEEINPL, CEEBETBL, CEEBLLST, and CEESG010. Refer to the program initialization and termination information in *z/OS Language Environment Vendor Interfaces* for details about the format of CEEMAIN and CEESTART.

Using XPCFTCH for a C/C++ routine

When the CICS global user exit XPCFTCH is used to intercept a C/C++ routine by providing an alternate entry point, CICS uses the address returned in `ceecics_rumentry_real` if it is available. It is supplied to XPCFTCH as the new PCUE_REAL_ENTRY parameter. You can also provide a MAIN that conforms to Language Environment with the hexadecimal value `X'47F0F014'` followed by `x'01'CEE`, DSASIZE, and the offset to the PPA1. A PPA1 is also required. Refer to the common interfaces and conventions information in *z/OS Language Environment Vendor Interfaces* for details about routine layout.

Note: CICS TS 3.1 and higher supports XPLINK programs in a CICS environment. The XPCFETCH user exit is not supported for XPLINK programs.

Creating a load notification user exit

The load notification user exit provides customers who are running applications with LRR active the ability to improve performance by preventing the use count for frequently used modules from dropping below one.

See *z/OS Language Environment Programming Guide* for more information about load notification user exit.

To create a load notification user exit:

1. Create an assembler language routine that conforms to the syntax described in *z/OS Language Environment Programming Guide*.
2. Assemble and link-edit your exit into a library that Language Environment can access at runtime, such as CEE.SCEERUN.
3. Code a CEEBLNUE CSECT that contains a CEEXLRT macro identifying your exit. The macro specifies your routine as a load notification user exit. The CEEBLNUE CSECT can be found in source file CEE.SCEESAMP(CEEBLNUE). See “CEEBLNUE CSECT” for more information.
4. Replace the existing CEEBLNUE CSECT with the updated CEEBLNUE as described in the following sections.

Identifying the load notification user exit

Use the CEEWLNUE sample job to specify your own load notification user exit.

Steps for modifying the JCL for CEEWLNUE

Perform the following steps to modify the JCL for CEEWLNUE:

1. Replace the comment lines following the ++ SRC statement in CEEWLNUE with your updated CEEBLNUE CSECT identifying your abnormal termination exit routine.

2. Change #GLOBALCSI to the data set name of your global CSI data set.

3. Change #TZONE to the name of your target zone.

4. Add necessary SMP/E PRE information for PTFs you have installed on your system which contain the same part.

When you are done, CEEWLNUE should run with a condition code of 0.

CEEBLNUE CSECT

CEEBLNUE is a CSECT explicitly linked with Language Environment, and it is the CSECT that you create by coding the CEEXLHD, CEEXLRT, and CEEXLST macros. Specifically, CEEBLNUE is linked with the CEEPLPKA module. CEEBLNUE CSECT is created through the use of the following Language Environment-provided assembler macros:

CEEXLHD

Defines the head of the list. CEEXLHD generates the CSECT statement and any header information required. It has no operands.

CEEXLRT

Identifies the name of the exit to register. Only one name can be provided

since only one load notification user exit may be registered. CEEXLRT has only one keyword parameter, LOADXIT=, which is the load name for the load notification user exit. There is a limit of 8 characters for the load name, and no validation of the name is performed by the macro.

CEEXLST

Defines the end of the list. CEEXLST generates the trailer for the CEEBLNUE load notification user exit CSECT. It has no parameters.

Language Environment validates the format of the CEEBLNUE CSECT and issues a load of the name as identified in the table. The LOAD is attempted only during region initialization when Library Routine Retention (LRR) is active. If the LOAD is successful, the exit is called for initialization according to the interface described in the following sections. If the LOAD is successful, the exit is registered and called during region initialization, after each successful load, and during region termination. This allows a STEPLIB to either contain or omit the load names.

Only one load notification user exit may be registered.

CEEBLNUE sample

Figure 4 shows the source for the IBM-supplied CEEBLNUE CSECT. It is provided in the CEE.SCEESAMP data set.

```

*/*****
*/
*/*   LICENSED MATERIALS - PROPERTY OF IBM   */
*/*                                         */
*/*   5645-001 5688-198                       */
*/*                                         */
*/*   (C) Copyright IBM Corp. 1991, 1997     */
*/*                                         */
*/*   All Rights Reserved                     */
*/*                                         */
*/*   US Government Users Restricted Rights - Use, duplication or */
*/*   disclosure restricted by GSA ADP Schedule Contract with IBM */
*/*   Corp.                                    */
*/*                                         */
*/*   Status = HMWL810                         */
*/*                                         */
*/*****
      CEEXLHD          ,User exit header
*-----*
*
*   To specify a load notification user exit,
*   change the line where CEEXLRT is specified,
*   by doing the following:
*
*   1. Change XXXXXXXX to the name of your load notification
*      user exit module name. This name must not be longer
*      than 8 characters.
*
*   2. Change the asterisk (*) in column 1 to a blank.
*
*-----*
*   CEEXLRT LOADXIT=XXXXXXXX
*   CEEXLST          ,Terminate the list

```

Figure 4. Sample of CEEBLNUE load notification user exit CSECT

Creating a storage tuning user exit

The storage tuning user exit provides a programming interface that allows you to collect Language Environment storage tuning information and to set the Language Environment runtime option values for STACK, LIBSTACK, HEAP, ANYHEAP and BELOWHEAP. See “Storage tuning user exit” on page 198 for more information.

The storage tuning user exit is available on CICS and on non-CICS environments when LRR is used.

To create a storage tuning user exit on CICS:

1. Create an assembler language routine that conforms to the syntax described in “Storage tuning user exit” on page 198.
2. Translate your exit with the CICS translator. The SYSEIB translator option must be used.
3. Assemble and link edit your exit into a library that is in the CICS DFHRPL DD concatenation. The member name of the exit must be CEECSTX.
4. Define program CEECSTX to CICS with LANGUAGE(ASSEMBLER). The definition for the program must be available at CICS start-up.

To create a storage tuning user exit on non-CICS:

1. Create an assembler language routine that conforms to the syntax described in “Storage tuning user exit” on page 198.
2. Assemble and link edit your exit into a library that Language Environment can load at runtime. The member name of the exit must be CEEBSTX.

Note: CICS TS 3.1 and higher supports XPLINK programs in a CICS environment. The non-CICS storage tuning exit (CEEBSTX) is used for these programs.

Chapter 8. Customizing the cataloged procedures

You can tailor the cataloged procedures supplied with Language Environment to suit the needs of your site. The procedures are part of the SCEEPROC cataloged procedure library.

You can tailor any of the following:

- If your site uses a prefix other than the IBM-supplied one, you can modify the data set name prefixes by using the LIBPRFX parameter.
- If you place CEE.SCEERUN and CEE.SCEERUN2 in the LNKLSTxx concatenation during installation, remove the STEPLIB DD statements.

Note: Since SCEERUN2 contains module names that do not intersect with any pre-Language Environment runtime library or any existing library, IBM recommends that SCEERUN2 be added to the LNKLST. This will not result in any adverse effects.

- If most of the programs at your site require a larger region for successful execution, change the default region size for the GO steps.
- Change UNIT=SYSDA in CEEWL, CEEWLG, AFHWL, AFHWLG, AFHWN, AFHWRL, and AFHWRLG.
- Tailor your TSO/E LOGON procedure. If you plan to run Language Environment applications under TSO/E, add SCEERUN and SCEERUN2 to the STEPLIB DD of the LOGON procedure, or use the TSO/E command TSOLIB to allocate SCEERUN and SCEERUN2; this is unnecessary if you place SCEERUN and SCEERUN2 into the LNKLST concatenation during installation.
- For programs that require the Language Environment Prelinker Utility, see *z/OS Language Environment Programming Guide* and *z/OS XL C/C++ Programming Guide* for details on changes to link-edit procedures. The requirement to use the Prelinker has been eliminated because the Binder directly supports input from the Language Environment conforming compilers. By choosing to eliminate usage of the Prelinker, the executable program will be a program object and must reside either in a PDSE or a UNIX file system.

Making the cataloged procedure library available to your jobs

Language Environment is shipped with a procedure library, CEE.SCEEPROC, that contains several procedures that can be used during application development with Language Environment. These procedures are summarized in Table 14:

Table 14. Language Environment invocation procedures in CEE.SCEEPROC

Procedure	Purpose
AFHWL	Link-edit a Fortran program.
AFHWLG	Link-edit and run a Fortran program.
AFHWN	Change any external names in conflict between C and Fortran to the Fortran-recognized name.
AFHWRL	Separate the nonshareable and shareable parts of a Fortran object module, and link-edit.
AFHWRLG	Separate the nonshareable and shareable parts of a Fortran object module, link-edit, and execute.

Table 14. Language Environment invocation procedures in CEE.SCEEPROC (continued)

Procedure	Purpose
CEEWG	Load and run a non-XPLINK Language Environment-conforming application.
CEEWL	Link-edit a non-XPLINK Language Environment-conforming application.
CEEWLG	Link-edit and run a non-XPLINK Language Environment-conforming application.
CEEXL	Link-edit an XPLINK Language Environment-conforming application.
CEEXLR	Link-edit and run an XPLINK Language Environment-conforming application.
CEEXR	Load and run an XPLINK Language Environment-conforming application.
EDCGNXLT (alias of EDC4P006)	Read a genxlt file and produce the translation table which is stored in the nominated LOADLIB.
EDCICONV (alias of EDC4P007)	Convert the characters from the input file from a coded character set definition to another character set definition and write the characters to the output file.
EDCLIB (alias of CRTCP002)	Maintain a C/C++ object code library.
EDCPL (alias of EDC4P002)	Prelink and link-edit a C/C++ application.

There are three ways to make the procedures available to your jobs. The method you choose depends on the special requirements and policies at your site. Use Table 15 to choose which method to use at your site.

Table 15. Deciding how to make cataloged procedures available to your jobs

If	Then	Result
You plan to use the IBM-supplied defaults and install into the default private procedure library.	Modify the JES2 start procedure.	Makes all procedures in the libraries available to any job in the system.
You are not using all the defaults and you want to choose which of the procedures to make available to general users.	Copy the procedures into a system or private PROCLIB.	Makes the procedures available to your installation jobs.
You are not using all the defaults.	Use the procedures as inline procedures.	Inserts the appropriate procedure into each job.

The process is as follows:

1. **Modify the JES start procedure.** You can do either of the following tasks:
 - Add a new //PROCnn DD statement for the Language Environment procedure library, CEE.SCEEPROC.
 - Concatenate the procedure library to the //PROC00 DD statement.

While testing, you can use the /*JOBPARM statement with the PROCLIB= parameter to make sure that your jobs use procedures from the correct library. To learn how to do this, see the section on JES2 control statements in *z/OS MVS JCL Reference*.

All procedures in the libraries that are added to the JES2 start procedure are available to any job in the system. The JES2 procedure is usually member JES2 in SYS1.PROCLIB.

2. **Place cataloged procedures in a system or private PROCLIB.** Copy the system procedures from the default libraries into an already-cataloged procedure library. You can use SYS1.PROCLIB as your cataloged procedure library. The copied procedures are callable by your installation jobs. However, procedures copied into a PROCLIB outside of SMP/E control are more difficult to maintain.

You can use the JCLLIB statement to specify a private PROCLIB. Do this by including the following statement after the JOB card and before the first EXEC statement in the job: `//PROCLIB JCLLIB ORDER=(CEE.SCEEPROC)`

3. **Use cataloged procedures as inline procedures.** Modify the procedure to reflect the high-level qualifiers you are using for the installation, and save your changes. Edit each job before you submit it, and copy the procedure into the job (inline).

Be sure to place `// PEND` at the end of the inline procedure.

Tailoring the cataloged procedures and CLISTs to your site

Several cataloged procedures and CLISTs are supplied with Language Environment and the Language Environment-conforming compilers. Some of these contain data set names that you may need to customize to your installation.

For information to help you customize the Language Environment cataloged procedures, see the topic discussed in Chapter 8, “Customizing the cataloged procedures,” on page 157 and the list in Table 14 on page 157.

For a list of names and possible modifications of CLISTs and all other cataloged procedures, see Table 16.

Several Fortran and C library routines have identical names. To correctly run existing Fortran applications under Language Environment, it is necessary to resolve all name conflicts. The Language Environment interface validation exit is a routine that automatically resolves conflicting library routine references within Fortran routines.

If the possibility exists of bringing in a Fortran routine when link-editing, activate the binder interface validation exit by modifying each of the cataloged procedures in Table 16 that performs a link-edit step to add an LKED parm of `EXITS(INTFVAL(CEEPINTV))`, and provide the following DD statement in the same step:

```
//STEPLIB DD DSN=CEE.SCEELKED,DISP=SHR
```

For more information about resolving conflicting names, see *z/OS Language Environment Programming Guide*.

Table 16. Cataloged procedures and CLISTs information

Category	Procedure names	Possible modifications
C/C++ cataloged procedures	Procedures found in hlq.SCCNPRC data set.	Modify the procedures to use the release of Language Environment you are using.

Table 16. Cataloged procedures and CLISTs information (continued)

Category	Procedure names	Possible modifications
COBOL cataloged procedures	IGYWC IGYWCG IGYWCL IGYWCLG IGYWCPG IGYWCPPL IGYWCPPLG IGYWPL	Modify the procedures to use the release of Language Environment you are using.
PL/I cataloged procedures	IEL1C IEL1CG IEL1CL IEL1CLG	Modify the procedures to use the release of Language Environment you are using.
Language Environment CLISTs	CMOD CPLINK C370LIB GENXLT ICONV DLLRNAME	<ul style="list-style-type: none"> • If you are not using the IBM-supplied default data set prefix, change the data set prefix symbolic parameter in all CLISTs. • Change parameters in CLISTs to match values at your site. • These procedures can be found in the CEE.SCEECLST data set.

Chapter 9. Using Language Environment under CICS

To make sure that CICS can communicate with Language Environment:

- Add the Language Environment required program resource definitions to the CICS System Definition (CSD) file.
- Ensure the required transient data (TD) queue resource definitions are defined to CICS.

Note: If the resource definitions are already defined in the CSD by the CICS utility, you should ensure they are not removed from the CICS group list used at startup.

- Add the Language Environment Library data sets to the CICS startup job stream.

Add program resource definitions for CICS

Update the CICS system definition (CSD) file using the program definitions in the CEECCSD member in the Language Environment sample (SCEESAMP). This member contains the necessary input to the CSD file utility program to define the Language Environment library routines to the CSD. The CSD group list that is used during CICS startup must include the CSD group that is associated with the Language Environment library routines. The group name for Language Environment routines is CEE in the sample CEECCSD.

For COBOL users, the OS/VS COBOL library routines (ILBOs) in Language Environment library SCEERUN are loaded by the operating system and do not require entries in the CSD.

The XPLINK program definitions in the CEECCSDX member, in the Language Environment sample (SCEESAMP) must be used to update the CICS system definition (CSD) file. Use the CEECCSDX member in addition to the CEECCSD member.

If you plan to run with program autoinstall and use the Language Environment CLER transaction, you must define the following statements using the CEDA transaction:

```
DEFINE PROGRAM(CEL4RTO) GROUP(CEE) LANGUAGE(ASSEMBLER)
EXECKEY(CICS)
DEFINE MAPSET(CELCLEM) GROUP(CEE)
DEFINE MAPSET(CELCLRH) GROUP(CEE)
DEFINE TRANS(CLER) PROG(CEL4RTO) GROUP(CEE)
```

Tip: If you use program autoinstall, Language Environment event handler modules in the range CEEEV001-CEEEV017 that are present in the CEE.SCEERUN might load during CICS/LE initialization, depending on the definitions in the CICS CEECCSD member and the autoinstall program. You can remove program definitions from the CEECCSD member to prevent them from being loaded during CICS/LE initialization. However, if autoinstall is active, the missing definitions from the CEECCSD are loaded dynamically unless the autoinstall program is modified to bypass any Language Environment modules that you do not want loaded.

To prevent this from occurring, you should start CICS with PGAIPGM=INACTIVE in the CICS SIT. To take advantage of the program autoinstall feature, you can create a PLTPI program to perform a CICS SET SYSTEM PROGAUTOINST (auto_active/cvda) to enable the feature for use later on in initialization. The URM can also be set in the same PLTPI program through the same SET SYSTEM command with the PROGAUTOEXIT parameter. If you do want to run with the autoinstall program, you can modify the autoinstall exit program to bypass any CEEEV0* modules that you do not want loaded.

The following autoinstall exit sample demonstrates this procedure:

```
DFHPGADX CSECT
DFHPGADX AMODE 31
DFHPGADX RMODE ANY
DFHREGS ,
*
*       If there is no commarea, return
OC   EIBCALEN,EIBCALEN
BZ   RETURN0
*
*       Address the commarea
L    R2,DFHEICAP
USING PGAC,R2
*
*       Omit autoinstall for Language Environment modules
CLC  PGAC_PROGRAM(6),=C'CEEV0'
BE   RETURNDD
*
*       Add user specific code here
*
*       Set the return code to OK
RETURNOK DS   0H
MVI  PGAC_RETURN_CODE,PGAC_RETURN_OK
B    RETURN0
*
*       Branch to this label if you elect not to define
*       the program
RETURNDD DS   0H
MVI  PGAC_RETURN_CODE,
PGAC_RETURN_DONT_DEFINE_PROGRAM
*
RETURN0 DS   0H
EXEC CICS RETURN,
END   DFHPGADX
```

Table 17. Excluding programming language support under CICS

If you do not run	Exclude these program definitions from the CEECCSD sample job
COBOL applications under CICS	CEEV004, CEEV005, IIGZMSGT, all programs that start with IGZ
C/C++ applications under CICS	CEEV003, IEDCMSGT, all programs that start with EDC or CEU
PL/I applications under CICS (Also VA PL/I)	CEEV010, CEEV011, IIBMMSGT, all programs that start with IBM

Guideline: If you use autoinstall and want to exclude one or more languages using this technique, be sure to implement these changes in your autoinstall exit to prevent them from being added dynamically.

Note: C was named AD/Cycle C/370™ before C++ was added. The sample JCL used the nickname C/370 to refer to either Language Environment-enabled version.

Add destination control table (DCT) entries

The CEECDCT member in the SCEESAMP sample library contains the necessary input to create the transient data queues as extrapartition data queues.

Entries for the transient data queues used by Language Environment are required in the destination control table. Language Environment uses the following transient data queues:

- **CESE:** messages, dumps, and reports are written to this queue. Each record written to the CESE queue has a header with terminal ID, transaction ID, date, and time. This queue is also used by C/C++ for stderr output and by PL/I for stream output data.
- **CESO:** C/C++ stdout stream output is written to this queue. The definition for this queue is required only if you use C/C++. Each record written to the CESO queue has a header with terminal ID and transaction ID.
- **CIGZ:** COBOL side file support for CEEDUMPs and Debug Tool. The definition for this queue is required only if you run COBOL programs compiled with the SEPARATE suboption of the TEST compiler option and you want to process side files using the CICS Extrapartition Transient Data Queue (TDQ) interface. This is an input-only queue.

In order to use the COBOL side file support on CICS for COBOL programs compiled with the TEST(SYM,SEPARATE) compiler option, you must define a transient data queue with the name CIGZ. Do not specify a DD for the CIGZ transient data queue in your CICS startup job. The DD will be dynamically allocated and deallocated as needed.

The following example is the source that can be used to define CIGZ in the DCT:

```
IGZDBGIN DFHDCT TYPE=SDSCI,          COBOL Side File Support
          DSCNAME=IGZDBGIN,
          TYPEFLE=INPUT
CIGZ     DFHDCT TYPE=EXTRA,          COBOL Side File Support
          DESTID=CIGZ,
          DSCNAME=IGZDBGIN,
          OPEN=DEFERRED
```

Figure 5 illustrates the format for the output transient data queues.

ASA	Terminal ID	Transaction ID	sp	Timestamp YYYYMMDDHHMMSS	sp	Message
1	4	4	1	14	1	132

Figure 5. Format of an output transient data queue

ASA The American National Standard carriage-control character

Terminal ID

A 4-character terminal identifier

Transaction ID

A 4-character transaction identifier

sp A space

Timestamp

The date and time displayed in the same format as that returned by the CEELCT service

Message

The message identifier and message text

These queues can have intrapartition, extrapartition, or indirect destinations. The record length for the transient data queue CESE must be at least 161.

We recommend that you put the required Language Environment entries in the CSD as TDQUEUE resource definitions (introduced in the CICS Transaction Server for z/OS). The Language Environment TD queues are included in the CICS-supplied CSD group called DFHDCTG, which is added to the DFHLIST automatically when initializing or upgrading a CSD. The following are the Language Environment entries created in the DFHDCTG:

```
DEFINE TDQUEUE (CESO)          GROUP(DFHDCTG)
  DESCRIPTION(LE/370 OUTPUT QUEUE)
  TYPE(EXTRA)                  TYPEFILE(OUTPUT)
  RECORDSIZE(133)              BLOCKSIZE(137)
  RECORDFORMAT(VARIABLE)      BLOCKFORMAT(UNBLOCKED)
                               DDNAME(CEEOUT)
*
DEFINE TDQUEUE (CESE)          GROUP(DFHDCTG)
  DESCRIPTION(LE/370 ERROR QUEUE)
  TYPE(EXTRA)                  TYPEFILE(OUTPUT)
  RECORDSIZE(161)              BLOCKSIZE(165)
  RECORDFORMAT(VARIABLE)      BLOCKFORMAT(UNBLOCKED)
                               DDNAME(CEEOUT)
*
DEFINE TDQUEUE (CIGZ)          GROUP(DFHDCTG)
  DESCRIPTION(COBOL SIDE FILE INPUT QUEUE)
  TYPE(SDSCI)                  TYPEFILE(INPUT)
                               DDNAME(IGZDBGIN)
```

See *CICS Transaction Server for z/OS System Definition Guide* for information provided by CICS about installing Language Environment support.

Use the DFHDCT macro to define the entries for CESE, CESO and CIGZ.

In addition to defining the transient data queues in the DCT, you must make sure that there is a DD statement in the CICS startup job for the transient data queues.

Note: Do not specify a DD for the CIGZ TDQ. It will be dynamically allocated and deallocated as needed.

If you define the CESE and CESO transient data queues as separate extrapartition data queues, the following example shows what you would specify in your CICS startup JCL:

```
//CEEMSG DD DSN=CUSTOMER.CEEMSG,DISP=SHR
//CEEOUT DD DSN=CUSTOMER.CEEOUT,DISP=SHR
```

For more information about the DFHDCT macro and the definitions of the queues and associated buffers, see *CICS Transaction Server for z/OS System Definition Guide*.

CEEMSG	DFHDCT	TYPE=SDSCI, Language Environment messages, dumps, reports	
		DSCNAME=CEEMSG,	X
		BLKSIZE=165,	X
		RECSIZE=161,	X
		RECFORM=VARUNBA,	X
		TYPEFLE=OUTPUT,	X
		BUFNO=1	
CESE	DFHDCT	TYPE=EXTRA,	X
		DESTID=CESE,	X
		DSCNAME=CEEMSG	
CEEOUT	DFHDCT	TYPE=SDSCI, C/C++ STDOUT stream	X
		DSCNAME=CEEOUT,	X
		BLKSIZE=137,	X
		RECSIZE=133,	X
		RECFORM=VARUNBA,	X
		TYPEFLE=OUTPUT,	X
		BUFNO=1	
CESO	DFHDCT	TYPE=EXTRA,	X
		DESTID=CESO,	X
		DSCNAME=CEEOUT	
IGZDBGIN	DFHDCT	TYPE=SDSCI, COBOL Side File Support	X
		DSCNAME=IGZDBGIN,	X
		TYPEFLE=INPUT	
CIGZ	DFHDCT	TYPE=EXTRA, COBOL Side File Support	X
		DESTID=CIGZ,	X
		DSCNAME=IGZDBGIN	X
		OPEN=DEFERRED	

Note: Xs are in column 72.

Figure 6. Example of DFHDCT macro

When DFHDCT encounters the entry names CESE, CESO, CIGZ, CEEMSG, and CEEOUT, it might generate messages stating that queue names beginning with the letter C are reserved for CICS. It is normal to receive these messages, and they do not indicate errors.

Specifying the side file interface to be used

COBOL can use one of two interfaces to access side files during Debug Tool debugging and CEEDUMP processing:

- CICS Extrapartition Transient Data Queues (TDQs)
- Direct QSAM access through a CICS Task Related User Exit (TRUE)

By default, COBOL will use the TDQ interface (using CICS SPI and API function calls) to access side files. If you would prefer COBOL to use the new direct QSAM TRUE interface instead of the TDQ interface, you need to enable the direct QSAM TRUE interface.

To enable the direct QSAM TRUE interface, specify the following INITPARM in your CICS startup parameters:

```
INITPARM=(DFHLETRU='USEQSAM')
```

Note that by providing this INITPARM, the direct QSAM TRUE interface will be used for:

- COBOL side files
- Debug Tool files such as listing, source, preference, USE, and LOG files

In order to use the direct QSAM TRUE interface, you need the PTFs for the following APARs to be applied to the appropriate products:

- CICS Transaction Server 3.1 PK67329
- CICS Transaction Server 3.2 PK68401
- COBOL component of Language Environment PK71852
- Debug Tool 8.1 PK69617
- Debug Tool 9.1 PK72833

Add Language Environment-CICS data sets to the CICS startup job stream

Before running any CICS transactions under Language Environment, you must add Language Environment to the startup job stream. *CICS Transaction Server for z/OS System Definition Guide* describes the CICS system startup procedure and provides an example of a CICS startup job stream.

To add the Language Environment-CICS data sets to CICS:

- Update the DFHRPL DD concatenation.

Add the Language Environment runtime library SCEERUN in the DFHRPL DD concatenation of the job that is used to start CICS.

If you are running COBOL programs on CICS, you must also add Language Environment runtime library SCEECICS in the DFHRPL DD concatenation. The SCEECICS library must be concatenated before the SCEERUN library.

Any libraries that contain runtime routines from earlier versions of COBOL, PL/I, and C/C++ should be removed from the DFHRPL DD concatenation.

If you are running COBOL V5.1 (or later) programs, you must also add the Language Environment runtime library SCEERUN2 in the DFHRPL DD concatenation of the job that is used to start CICS.

If you are running COBOL V5.1 (or later) programs that were compiled with the TEST compiler option on CICS, you must also add system libraries SYS1.MIGLIB and SYS1.SIEAMIGE in the DFHRPL DD concatenation.

- If SCEERUN is not in LNKST or LPALST, then you must include the correct Language Environment routines into an authorized library that is part of the STEPLIB DD concatenation in the CICS startup job. If SCEERUN is in LNKST/LPALST, then you do not have to add SCEERUN to the STEPLIB DD concatenation in the CICS startup job. You can either:

1. Authorize the Language Environment runtime library SCEERUN and then include it in the STEPLIB DD concatenation in the CICS startup job. (The SCEERUN2 data set does not need to get added to this concatenation.)
2. Put only those Language Environment routines that are needed by CICS using the STEPLIB into another library.

If you use the second method, you must make the following Language Environment routines available by using the STEPLIB:

- CEECCICS, CEECTCB
- IGZCWTO: Used for COBOL support.
- IGZCMTUE: Used for COBOL support.
- IGZIDYN: Used for COBOL support.
- ILBO routines: If you are running OS/VS COBOL programs, all of the ILBO routines must be available.

Remove any libraries that contain runtime routines from earlier versions of COBOL and C/370 from the STEPLIB DD concatenation.

Note:

1. The previously mentioned library routines required from the STEPLIB might also be available by using the JOBLIB or the LNKSTnn member.
2. There is no CICS startup option for Language Environment. If CICS locates CEECCICS, it attempts to initialize Language Environment. If the modules have not been installed correctly, Language Environment initialization fails, and CICS generates an error message to that effect.

Language Environment automatic storage tuning for CICS

Language Environment automatic storage tuning for CICS provides automatic storage tuning (AUTOTUNE) of Language Environment STACK, LIBSTACK, HEAP, BELOWHEAP and ANYHEAP initial size values. Automatic storage tuning of the Language Environment storage areas can improve the performance of applications running on CICS by reducing the CICS GETMAIN/FREEMAIN activity associated with acquiring Language Environment stack and heap increments. In order to use Language Environment automatic storage tuning for CICS, the CICS system initialization parameter AUTODST must be set to YES. The CICS system initialization parameter AUTODST is available only on:

- CICS Transaction Server Version 1 Release 3 with APARs PQ39052, PQ45031, and PQ55351.
- CICS Transaction Server Version 2.
- CICS Transaction Server Version 3.

Note: When Language Environment Automatic Storage Tuning for CICS is used, the capability of the storage tuning user exit is changed. For example, the storage tuning user exit can no longer get storage information. See *z/OS Language Environment Programming Guide* for information about the Language Environment storage tuning user exit.

Note: CICS TS 3.1 and higher supports XPLINK programs in a CICS environment. The automatic storage tuning exit is not supported for these programs.

Enclaves eligible for automatic storage tuning

When running with Language Environment automatic storage tuning for CICS, the actual storage tuning is performed for Language Environment enclaves when one of the following conditions is met:

- The main program is not link-edited with a CEEUOPT.
- The main program is link-edited with a CEEUOPT, and the CEEUOPT does not specify values for any of the following runtime options: STACK, LIBSTACK, HEAP, BELOWHEAP or ANYHEAP.

Note:

1. A CEEUOPT is present in C/C++ main programs that use #pragma runopts when one of the following compilers were used: z/OS XL C/C++, OS/390 C/C++, C/C++ Compiler for MVS/ESA, or AD/Cycle C/370.
2. A CEEUOPT is present in PL/I main programs that use PLIXOPT when one of the following compilers are used: Enterprise PL/I or PL/I for MVS & VM.

Automatic storage tuning behavior

Automatic storage tuning values are managed for each load module that is used to start an enclave for Language Environment. For example, transaction ATMW starts program COBOLA (which starts an enclave for Language Environment). COBOLA does a CICS LINK to COBOLB which starts another Language Environment

enclave. COBOLB does a dynamic call to COBOLC (when a dynamic call is done, we are still running in the same enclave). In this example, automatic storage tuning is done for the enclaves started for COBOLA and COBOLB.

When running with Language Environment automatic storage tuning for CICS, Language Environment continuously monitors the amount of Language Environment storage allocated in the enclave for STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP. The enclave ends normally, Language Environment will automatically increase the initial size values for STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP as determined by the amount of storage allocated.

In more detail, Language Environment automatic storage tuning for CICS behaves as follows:

- When a main program starts a Language Environment enclave the first time in a CICS region and the enclave is eligible for automatic storage tuning, Language Environment uses the values for STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP from the normal search order for runtime options. When a main program starts an eligible enclave a subsequent time, Language Environment uses the initial sizes for STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP as determined by automatic storage tuning.
- Whenever a Language Environment enclave is initialized and it is eligible for automatic storage tuning, Language Environment collects the total amount of storage allocated for STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP while the enclave is active.

Note: When Language Environment automatic storage tuning for CICS is used, Language Environment collects the amount of storage allocated. It does not collect the amount of storage used.

- When the enclave ends with an unhandled condition, Language Environment does not update the automatic storage tuning values. When the enclave ends normally, Language Environment automatic storage tuning increases the initial size for STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP to the amount of storage allocated only when the amount of storage allocated is larger than the initial size. The next time the main program initiates a Language Environment enclave, Language Environment automatic storage tuning will use the updated initial size values.
- Language Environment automatic storage tuning never decreases the initial size values.

Altering the automatic storage tuning behavior

To alter the behavior of the Language Environment automatic storage tuning for CICS, the storage tuning user exit can be used. For example, the storage tuning user exit can be used as follows:

- To apply its own logic and determine which programs are eligible for automatic storage tuning.
- To set limits on the initial sizes used by Language Environment automatic storage tuning for CICS.

See “Storage tuning user exit” on page 198 for information about Language Environment storage tuning user exit.

Chapter 10. Using Language Environment under IMS

If you are running programs that require Language Environment in an IMS/TM dependent region, such as an IMS message processing region, you can improve performance if you use Language Environment library routine retention.

With library routine retention in effect, Language Environment keeps certain resources in memory when an application program ends, making subsequent invocations of programs that use Language Environment much faster because the Language Environment resources left in memory are reused.

Following is a partial list of the resources Language Environment keeps in memory with library routine retention in effect:

- Language Environment runtime load modules
- Language Environment storage associated with the management of the runtime load modules
- Language Environment storage for start-up control blocks

Initializing library routine retention

To use Language Environment library routine retention in an IMS dependent region, you must do the following:

1. In your JCL or procedure used to bring up IMS dependent regions, specify that you want IMS to invoke dependent region preinitialization routines. Do this by specifying a suffix on the PREINIT keyword of the IMS dependent region procedure.
2. In the DFSINTxx member of IMS.PROCLIB (where xx is a suffix specified by the PREINIT keyword), include the name CEELRRIN or CEELRRXP.

When IMS invokes the module CEELRRIN or CEELRRXP, Language Environment library routine retention is initialized.

Note: The source for module CEELRRIN or CEELRRXP is available in the SCEESAMP library in member CEELRRIN or CEELRRXP. If this source does not meet your needs, you can create your own assembler program to initialize Language Environment library routine retention. If you create your own load module to initialize Language Environment library routine retention, you need to put the name of the module in the DFSINTxx member.

Ending library routine retention

Language Environment provides a routine called CEELRRTR to terminate library routine retention. However, this routine does not need to be used when running on IMS/TM. If library routine retention is initialized, and the IMS Program Control Task is terminated (for example, due to an ABEND), the operating system will free the Language Environment resources as part of task termination. Then when the IMS Program Control Task is reattached, the preinitialization routines get control before IMS scheduling is attempted.

For more information about specifying IMS dependent region preinitialization routines, see *IMS/ESA[®] Customization Guide*. For more information about Language Environment library routine retention, see *z/OS Language Environment Programming Guide*.

Chapter 11. Customizing language-specific features

In addition to tailoring your Fortran LIBPACKs, you might want to customize COBOL, C/C++, Fortran, and PL/I features in order to tune or diagnose the performance of Language Environment for your site.

Restriction: You cannot customize these features for AMODE 64 applications.

Choices to make now

First, decide which language-specific features you should modify for your site. For more information about the C/C++, Fortran, and PL/I features you can customize, see:

- Appendix B, “Using Fortran with Language Environment,” on page 217
- *PL/I for MVS & VM Compiler and Run-Time Migration Guide*

You also need to choose which sample customization jobs you will need to modify and run. Table 18 lists the sample jobs provided on the distribution tape to help you customize COBOL, C/C++, Fortran, and PL/I features. These jobs are part of Language Environment sample library SCEESAMP.

Table 18. Customizing programming languages with sample customization jobs

To	Use this sample job
Modify the OS/VS COBOL compatibility library routines	IGZWZAP
Modify the COBOL runtime environment	IGZWARRE
Customize the parameter list processing when a COBOL program is invoked with an ATTACH SVC on z/OS	IGZWAPSX
Customize the C/C++ locale time information	EDCLOCL
Relink OS PL/I Version 2 shared library and OS PL/I Version 1 CICS or tasking shared library	IBMRLSLA
Relink OS PL/I Version 1 non-CICS and non-tasking shared library	IBMRLSLB
Tailor the Language Environment Fortran unit attribute table	AFHWEUAT
Tailor the VS FORTRAN compatibility unit attribute table	AFHWVUAT
Tailor the VS FORTRAN compatibility runtime options defaults	AFHWVPRM
Tailor the VS FORTRAN compatibility Error Option Table	AFHWVOPT

Modifying the OS/VS COBOL compatibility library routines

Use the IGZWZAP sample job to modify the OS/VS COBOL compatibility library routines. The job lets you apply superzaps to make Language Environment COBOL behave like OS/VS COBOL. See Table 19 on page 172 for a summary of the modifications you can make with the job. “OS/VS COBOL considerations” on page 172 explains the superzaps in detail.

Table 19. Using the usermods in the IGZWZAP job to modify the COBOL compatibility library

Usermod	Contains superzaps to	For
IGZWZA1	Continue to force USER ABEND 0100, 0201, 0303, or 0304 and message IFK302I	Certain error situations during VSAM file processing
IGZWZA2	Force USER ABEND 0295	A serious error detected at runtime
IGZWZA3	Add A, B, and E as valid numeric signs	The IF NUMERIC CLASS TEST

To modify the JCL for IGZWZAP:

1. Change #GLOBALCSI to the data set name of your global CSI data set.
2. Change #TZONE to the name of your target zone.

IGZWZAP should run with a condition code of 0.

OS/VS COBOL considerations

If the COBOL programmers at your site are familiar with OS/VS COBOL, you may want to modify Language Environment COBOL to make it behave like the OS/VS COBOL runtime. The IGZWZAP member is a sample job provided in CEE.SCEESAMP to apply USERMODs IGZWZA1, IGZWZA2, and IGZWZA3, which are described in the following sections. For instructions on modifying the JCL for the IGZWZAP job, see “Modifying the OS/VS COBOL compatibility library routines” on page 171.

User modifications for the OS/VS COBOL library also apply for the OS/VS COBOL compatibility library routines.

VSAM considerations

Support for VSAM processing in OS/VS COBOL Release 2 and in the OS/VS COBOL compatibility library routines is consistent with the I/O language specified in the COBOL standard, American National Standard COBOL, X3.23-1974. However, OS/VS QSAM and VSAM support in OS/VS COBOL Release 1 is not consistent with the standard.

File status

The FILE STATUS clause is optional. Specifying FILE STATUS for a VSAM file lets you monitor the status of the file's I/O operations by testing the FILE STATUS values. Code the FILE STATUS clause for all appropriate files and test the FILE STATUS (status key) after each input/output statement, including the OPEN statement. FILE STATUS detects error conditions so you can handle them before processing continues.

If you do not specify FILE STATUS and test for the appropriate status key values, you might get undetected errors and erroneous program results.

User abends

In certain error situations during VSAM file processing, Release 1 of the OS/VS COBOL library modules forced user abends during program execution. OS/VS COBOL Release 2 support eliminated four of these user abends. In place of the abends, a FILE STATUS value is set when an I/O operation fails, and execution continues.

Status key values are set to 90, 93, 95, or 95 rather than the forced USER ABEND 0100, 0201, 0303, or 0304, respectively. The program should test the

status key value after each I/O operation to make sure its successful completion. OS/VS COBOL Release 2 support also no longer issues the object-time message 'IKF302I'. In place of this message, the FILE STATUS is set to a value of 30.

Because some users might depend on the previous abends and message, you can apply superzaps as user modifications to continue to force USER ABEND 0100, 0201, 0303, or 0304, and continue to force message IKF302I. The IGZWZA1 USERMOD in the IGZWZAP sample job contains the superzaps to do this.

JOB STEP ERROR COMPLETION CODE (RC12/ABEND U0295)

In OS/VS COBOL, if a COBOL library subroutine detects a serious error at execution time (for example, a SYSOUT DD statement is missing), ILBOSRV1 sets the return code and the JOB STEP COMPLETION/ CONDITION CODE to 12 (CC12) upon terminating the run unit. A return code of 12 is compatible with versions 2 and 3 of American National Standard COBOL.

If you want to change the default return code, you can overlay the halfword X'000c' at displacement X'0002' into CSECT ILBOSRV with the error completion code of your choice. If the halfword is set to a NEGATIVE value during STOP RUN or GOBACK processing, the program is terminated with the USER ABEND 0295 (ABENDU0295) instead of a return code 12.

Because some users might depend on programs abending in the preceding conditions, you can apply the superzap as a user modification (IGZWZA2) to force a USER ABEND 0295.

IF NUMERIC CLASS TEST allows only C, D, and F

A, B, and E were valid signs for an IF NUMERIC compare in OS/VS COBOL Release 1, but the current release allows only C, D, and F as valid signs for an IF NUMERIC compare. Because some users might depend on the COBOL NUMERIC CLASS TEST, which includes A, B, and E as valid numeric signs, you can apply a provided superzap (IGZWZA3) as a user modification to add A, B, and E as valid numeric signs.

In any case, incorrect data in a data item used for a numeric class comparison is accepted as valid if its hexadecimal notation contains a valid sign. (For example, EBCDIC 'A', or X'C1', is a valid numeric sign for external decimal; and EBCDIC '%', or X'6C', is a valid numeric sign for internal decimal.)

Modifying the COBOL parameter list exit

The COBOL parameter list exit routine IGZEPSX can be modified to alter the parameter list processing when a COBOL main program is invoked by an z/OS ATTACH. This exit is ignored by programs compiled with COBOL V5R1 and later releases.

With the IBM supplied default COBOL parameter list exit, if the COBOL main is invoked by using the ATTACH SVC, a halfword-prefixed string is passed to the application after runtime options have been removed. The source of this string is dependent on the environment in which the ATTACH is issued:

- If the ATTACH is issued by z/OS to invoke a batch program, the string is specified using the PARM field of the EXEC statement.

- If the ATTACH is issued by TSO/E to attach a Command Processor (CP), the string is specified as part of the command embedded within the CP parameter of the TSO/E ATTACH CP command.
- If the ATTACH is not issued by z/OS or TSO/E, the string is specified using the PARM field of the ATTACH macro.

If the default behavior does not meet your needs, the COBOL parameter list exit IGZEPSX can be altered to set the parameter list processing so that R1 and the parameter list is passed without change to the main COBOL program.

Use the IGZWAPXS sample job to change the COBOL parameter list exit. You must replace the comment in IGZWAPXS with your source for IGZEPSX. You can copy the source for the IBM-supplied default COBOL parameter list exit from IGZEPSX in SCEESAMP and modify it to suit your needs. Included in IGZEPSX is sample code that can be used to get the same parameter list processing that is done when running COBOL programs with the VS COBOL II runtime library.

Steps for modifying the JCL for IGZWAPXS

Perform the following steps to modify the JCL for IGZWAPXS:

1. Replace the comment lines following the ++ SRC statement in the job with your source program for the COBOL parameter list exit.

2. Change #GLOBALCSI to the data set name of your global CSI data set.

3. Change #TZONE to the name of your target zone.

When you are done, IGZWAPXS should run with a condition code of 0.

Modifying the COBOL runtime environment

The IGZERREO CSECT provides a method of specifying additional parameters for COBOL to use at runtime. These parameters are separate from the Language Environment runtime options and apply only to the COBOL runtime. These parameters can be used to change the behavior of the COBOL reusable environment, the behavior of nested enclaves in a reusable environment, and the amount of data dumped for BLL cells in a formatted dump.

The parameters for the COBOL runtime environment are specified through IGZERREO using the keyword parameters REUSENV, NESTENC, and DUMPBLL.

Modifying COBOL reusable environment behavior

The COBOL reusable environment behavior can be modified to control how program checks that occur in the non-Language Environment-conforming driver are handled, as well as to control whether COBOL programs can run in a nested enclave in the reusable environment. The COBOL reusable environment is established with the RTEREUS runtime option or a call to either ILBOSTPO or IGZERRE INIT.

With the IBM supplied default setting for COBOL's reusable environment behavior (IGZERREO with REUSENV=COMPAT), when a program check occurs while the reusable environment is dormant (for example, between a GOBACK from a top-level COBOL program to the non-Language Environment conforming

assembler driver and the next call to a COBOL program), a S0Cx abend will occur. This behavior is compatible with the VS COBOL II and OS/VS COBOL runtimes, but it significantly impacts the performance when an Enterprise COBOL for z/OS, COBOL for OS/390 & VM, or COBOL for MVS & VM program is invoked repeatedly in a COBOL reusable environment. The performance degradation is caused by Language Environment issuing an ESPIE RESET when the reusable environment becomes dormant and then an ESPIE SET upon reentering the reusable environment.

COBOL's reusable environment behavior can be modified (IGZERREO with REUSENV=OPT) so that all program checks will be intercepted by Language Environment, even those that occur while the reusable environment is dormant. In this case, a program check that occurs while the reusable environment is dormant will result in a 4036 abend from Language Environment. However, since Language Environment does not have to issue the ESPIE RESET and ESPIE SET between invocations of the COBOL program, this can be faster than using REUSENV=COMPAT.

Modifying nested enclave behavior

With the IBM-supplied default setting for COBOL's reusable environment behavior (IGZERREO with NESTENC=NO), when a reusable environment is active and a nested enclave is created that contains a COBOL program, COBOL will diagnose this with error message IGZ0168S.

COBOL's reusable environment behavior can be modified (IGZERREO with NESTENC=YES) so that a nested enclave containing a COBOL program will continue to run, even though a reusable environment is still active in the parent enclave.

- When you run a COBOL program in a nested enclave.
- The COBOL program is not part of the reusable environment.
- When the nested enclave ends, all the resources associated with the nested enclave are freed.

If a STOP RUN is done in the nested enclave, it only terminates the nested enclave, and does not terminate the COBOL reusable environment.

Modifying COBOL formatted dump behavior

With the IBM-supplied default, 4096 bytes of data are written for each BLL cell in active programs and no data for BLL cells in programs that are not active.

The COBOL runtime environment behavior can be modified to change how much data from each BLL is written to the CEEDUMP. The DUMPBLL parameter allows two suboption keywords, ACTIVE and INACTIVE, to specify the length of data from the BLL to be dumped for active and non-active programs respectively. Each suboption must be associated with a length value. The suboption keywords must be spelled out completely. If the suboption is specified, a length value must be specified. The length value must be between 0 and 4096. If the value specified is greater than 0, it must be a multiple of 32.

If a suboption is skipped entirely, the default value is used for that suboption. A partially specified suboption or a suboption with a keyword or length omitted is diagnosed as an error during the IGZRREOP macro processing. This causes a nonzero return code.

Modifying the behavior of the COBOL runtime environment

Use the IGZWARRE sample job to change the behavior of COBOL's runtime environment. You must modify the IGZRREOP macro invocation, depending on the function that you want.

To run with VS COBOL II and OS/VS COBOL runtime compatibility mode (that is, the user has control of program checks that occur when the COBOL runtime environment is dormant, resulting in an additional performance cost), use **IGZRREOP REUSENV=COMPAT**

To run with optimum performance (for example Language Environment intercepts all program checks that occur when the COBOL runtime environment is dormant and converts them to a 4036 abend, resulting in improved performance), use **IGZRREOP REUSENV=OPT**

To disable nested enclave support in the reusable environment, use **IGZRREOP NESTENC=NO**

To enable nested enclave support in the reusable environment, use **IGZRREOP NESTENC=YES**

To change the amount of data dumped for BLL cells in a CEEDUMP, use **IGZRREOP DUMPBLL=((suboption))**. Specify either or both of the suboption values (ACTIVE,nnnn) and (INACTIVE,nnnn). The value must be between 0 and 4096 and a multiple of 32 to replace nnnn.

Modifying the JCL for IGZWARRE

Perform the following steps to modify the JCL for IGZWARRE:

1. Copy the IGZERREO member from CEE.SCEESAMP into IGZWARRE in place of the comment lines following the ++ SRC statement.

2. Change the REUSENV NESTENC, and DUMPBLL parameters on the IGZRREOP macro statement to the desired value.

3. Change #GLOBALCSI to the data set name of your global CSI data set.

4. Change #TZONE to the name of your target zone.

IGZWARRE should run with a condition code of 0.

Modifying the COBOL debug file name

When a COBOL program is compiled with the SEPARATE suboption of the TEST compiler option, the file name of the separate debug file created by the COBOL compiler is stored in the object deck. The file name can be one of the following:

- A data set name
- A data set name with a member name
- Az/OS UNIX file name

At runtime, when a Language Environment-formatted dump is requested, the runtime gets the debug file name from the COBOL executable. If the debug file created at compile time is not available, the formatted dump does not format the local variables of the program.

If the COBOL debug files are kept in a file that is different from the file used at compile time, you can use the COBOL debug file user exit to provide a file name.

The COBOL debug file user exit also gets control when the debug tool is used to debug a COBOL program compiled with the SEPARATE suboption of the TEST compiler option.

The COBOL debug file user exit can be used in all environments. The user exit is called each time a new COBOL debug file is required. This gives the exit the opportunity to change the file name.

Using a COBOL debug file user exit

To use the COBOL debug file user exit in a non-CICS environment:

1. Write an assembler language routine that conforms to the interface of the COBOL debug file user exit as described in “Using the COBOL debug file user exit interface.”
2. Assemble and link edit your user exit into a load library that Language Environment can load at runtime. The member name of the user exit must be IGZIUXB.

To use the COBOL debug file user exit in a CICS environment:

1. Write an assembler language routine that conforms to the interface of the COBOL debug file user exit as described in “Using the COBOL debug file user exit interface.”
2. If your user exit has CICS commands, translate it with the CICS translator using the SYSEIB translator option.
3. Assemble and link edit your user exit into a load library in the CICS DFHRPL DD concatenation. The member name of the user exit must be IGZIUXC.
4. If not already done, define the IGZIUXC program to CICS. When you define the program to CICS you do not need to specify the language. However, if you do want to specify the language, you must specify LANGUAGE(ASSEMBLER).

Using the COBOL debug file user exit interface

The name of the COBOL debug file user exit is:

- IGZIUXB for non-CICS
- IGZIUXC for CICS

The COBOL debug file user exit is loaded the first time you need to use a COBOL debug file. If the load of the user exit is not successful, Language Environment does not issue a message and does not attempt to call the user exit.

Syntax

For IGZIUXB (non-CICS):

```
IGZIUXB(Interface_Version,  
        Name_Of_Debug_File,  
        Name_Of_CU)
```

For IGZIUXC (CICS):

```
IGZIUXB(Interface_Version,  
        Name_Of_Debug_File,  
        Name_Of_CU,  
        CICS_SYSTEM_EIB)
```

Where:

Interface_Version (INPUT)

Is a 4-byte integer with interface version. The value is 1.

Name_Of_Debug_File (INPUT/OUTPUT)

Is a halfword-prefixed 256-byte character string that has the name of the debug file. This is an input/output field. On input it contains the name of debug file name that was used at compile time. On output, the name length and name can be updated by the user exit. The new name information is used by the runtime when R15 is zero on return. The name is not padded with blanks on input. On output, the name length must reflect the length of the name without blanks.

Name_Of_CU (INPUT)

Is a halfword-prefixed 160-byte character string that has the name of the compile unit. The compile unit name of a program is the program name. The compile unit name of a class is the class name.

CICS_SYSTEM_EIB (INPUT)

Is the CICS system EIB (EXEC Interface Block)

CICS considerations

CICS commands can be used in the COBOL debug file user exit. However, the COBOL debug file user exit must adhere to the following conventions when using EXEC CICS commands: The COBOL debug file user exit must use the CICS system EIB with the SYSEIB translator option. The CICS commands must use the RESP option.

Register conventions

Register conventions for the COBOL debug file user exit are:

Table 20. Register conventions for the COBOL debug file user exit

Register	Description
1	Address of the parameter list
12	Address of the CAA
13	Address of a dynamic save area (DSA). The user exit routine can save the registers here across its processing
14	Contains the return address
15	Contains the entry point address upon entry and the return code upon exit

Usage notes

- The COBOL debug file user exit must be written in assembler language and must be reentrant. If you write the COBOL debug file user exit in Language Environment-enabled assembler, you must specify MAIN=NO on the CEEENTRY macro.

- The COBOL debug file user exit must not call any HLL programs. However, it can call other assembler routines.
- The COBOL debug file user exit must not create a Language Environment enclave.
- R15 must be set to zero upon return when the debug file name is changed. If R15 is nonzero, any change to the debug file name is ignored.
- Changes to the file name must be fully qualified. If the debug file is a PDS or a PDSE, the file name returned from the user exit must be the name of the PDS/PDSE along with the member name.
- The COBOL debug file user exit is called in AMODE(31) and must return in AMODE(31).

COBOL debug file user exit samples

Language Environment provides a sample COBOL debug file user exit for non-CICS and CICS environments.

- The sample COBOL debug file user exit for non-CICS is available in SCEESAMP in member IGZWIUXB.
- The sample COBOL debug file user exit for CICS is available in SCEESAMP in members IGZWIUXC and IGZWIUXD.
- The sample user exits take the member name used when storing the COBOL debug file at compile time. They look for the member in a concatenated PDS/PDSE under DD SYSDEBUG. If the member is found, the data set name and member are returned.
- **Restriction:** The sample user exits do not provide a new name if the debug file is stored in a sequential data set or a z/OS UNIX file at compile time.

Changing the C/C++ locale time information

Use the EDCLLOCL job to change the C/C++ locale time information for your site.

Recommendation: Do not install this usermod. The default C/C++ locale (EDC\$\$370) will by default obtain the time zone difference from Greenwich mean time from the system. If your C/C++ application requires a different time zone other than the one obtained from the system, you can use the `tzset()` and the TZ environment variable described in *z/OS XL C/C++ Runtime Library Reference*.

Modifying the JCL for EDCLLOCL

Perform the following steps to modify the JCL for EDCLLOCL

1. Change #GLOBALCSI to the data set name of your global CSI data set.

2. Change #TZONE to the name of your target zone.

When you are done, EDCLLOCL should run with a condition code of 0.

Part 3. Appendixes

Appendix A. Language Environment user exits

Language Environment provides support for the following user exits:

Assembler user exit

Performs functions for enclave initialization, normal and abnormal enclave termination, and process termination. See “Assembler and HLL user exits.”

High-level language (HLL) user exit

Performs functions for enclave initialization. See “Assembler and HLL user exits.”

Abnormal termination user exit

Collects problem determination data when Language Environment is terminating an enclave due to an unhandled condition. See “Abnormal termination exit” on page 193.

Load notification user exit

Improves performance by preventing frequently used modules from being loaded and deleted with each use. The load notification user exit is only available when Library Routine Retention (LRR) is used. See “Load notification user exit” on page 195.

Storage tuning user exit

Provides a programming interface that allows you to collect Language Environment storage tuning information and to set the Language Environment runtime option values for STACK, LIBSTACK, HEAP, ANYHEAP and BELOWHEAP. The storage tuning user exit is available on CICS and on non-CICS when LRR is used. See “Storage tuning user exit” on page 198.

Assembler and HLL user exits

IBM offers a default version of the CEEBXITA assembler user exit that you can customize during your Language Environment installation and use on a global or installation-wide basis. After installation, you can again customize CEEBXITA and link it directly to applications to use on an application-specific basis. The application-specific exit is used only when you run that application. In this case the installation-wide assembler user exit is not executed.

IBM also provides an HLL user exit, CEEBINT, that you can modify and use after installation. The HLL user exit is used during enclave initialization. Language Environment supplies an IBM-supplied default HLL user exit, or you can code one in C, PL/I, or Language Environment-conforming assembler language. You cannot write one in COBOL or Fortran.

After the enclave has been established, the HLL user exit is invoked and passed a parameter list that conforms to the Language Environment definition. The parameter list is described in *z/OS Language Environment Programming Guide*.

When assembler and HLL user exits are invoked

Figure 7 on page 184 shows the timing of the invocations of the assembler and HLL user exits at initialization and termination processing.

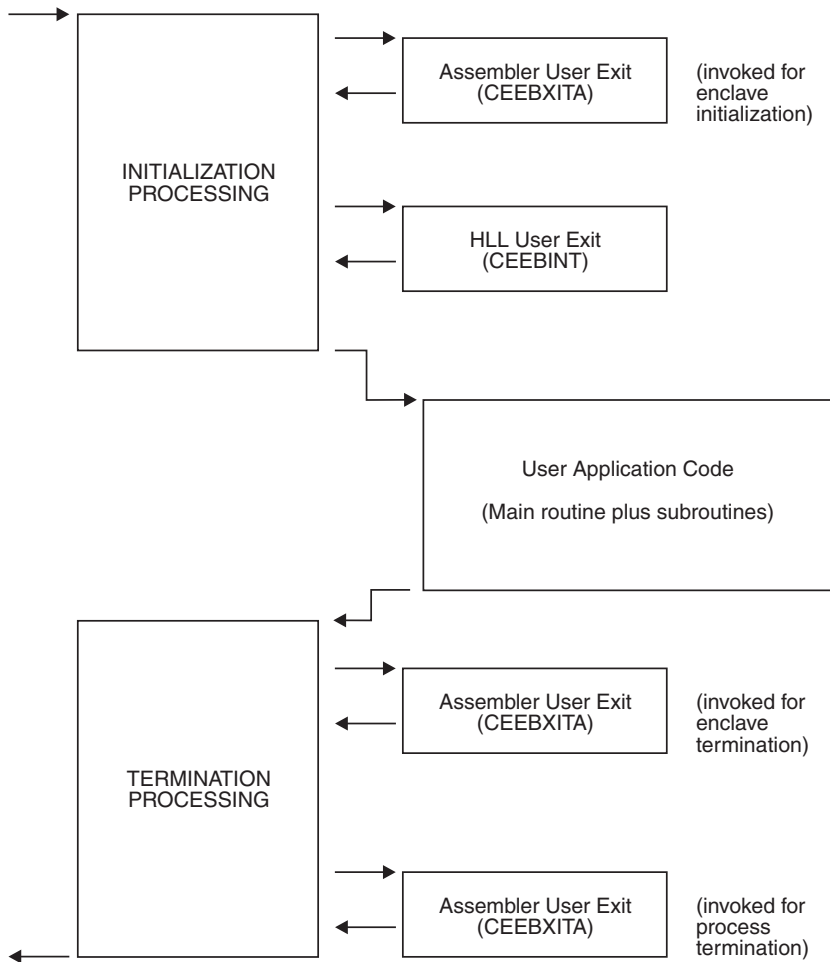


Figure 7. Location of user exits

In Figure 7, runtime user exits are invoked in the following sequence:

1. Assembler user exit is invoked for enclave initialization
2. Environment is established
3. HLL user exit is invoked
4. Main routine is invoked
5. Main routine returns control to caller
6. Assembler user exit is invoked for termination of the enclave
CEEBXITA is invoked for enclave termination processing after all application code in the enclave has completed, but before any enclave termination activity.
7. Environment is terminated
8. Assembler user exit is invoked for termination of the process
CEEBXITA is invoked again when the Language Environment process terminates.

Language Environment provides the CEEBXITA assembler user exit for termination but does not provide a corresponding HLL termination user exit.

CEEBXITA behaves differently, depending upon when it is invoked, as described in the following sections.

CEEEXITA behavior during enclave initialization

The CEEEXITA assembler user exit is invoked before enclave initialization is performed. You can use CEEEXITA to help establish your application runtime environment. For example, in the assembler user exit you can specify the stack and heap runtime options and allocate data sets. You can also use the user exit to interrogate program parameters supplied in the JCL and change them if you want. In addition, you can specify runtime options in the user exit by using the CEEAUE_A_OPTIONS field of the assembler interface.

z/OS considerations: Under z/OS, CEEEXITA returns control to Language Environment initialization.

CEEEXITA behavior during enclave termination

The CEEEXITA assembler exit is invoked after the user code for the enclave has completed, but before the occurrence of any enclave termination activity. In other words, the assembler user exit for termination is invoked when the environment is still active. For example, CEEEXITA is invoked before the storage report is produced (if you requested one), data sets are closed, and the debugger is invoked for enclave termination.

The z/OS assembler user exit permits you to request an abend. Under z/OS, including TSO and CICS, you can also request a dump to assist in problem diagnosis. Because termination activities have not yet begun when the user exit is invoked, the majority of storage has not been modified when the dump is produced.

You can request the abend and dump in the assembler user exit for all enclave-terminating events including:

- The situation that occurs in PL/I when the ON condition (including ERROR or FINISH) is raised and one of the following conditions is true:
 - The program does not have an appropriate ON-unit.
 - The ON-unit does not terminate with a GOTO.
 - The GOTO is not allowed.

This rule applies only to the conditions that cause termination of the program.

- Return from the main routine
- A Debug Tool QUIT command
- An HLL stop statement such as:
 - C exit()
 - COBOL STOP RUN
 - PL/I STOP or EXIT
 - Fortran STOP
- An unhandled condition of severity 2 or above

If a dump is requested in the user assembler exit and an unhandled condition has occurred, this dump will overwrite the dump taken by TERMTHDACT(UADUMP).

CEEEXITA behavior during process termination

The CEEEXITA assembler exit is invoked after:

- All enclaves have terminated
- The enclave resources have been relinquished
- Any Language Environment-managed files have been closed
- Debug Tool has terminated

At this time you can free allocated files and request an abend.

During termination, CEEBXITA can interrogate the Language Environment reason and return codes and, if necessary, request an abend with or without a dump. This can be done at either enclave or process termination.

Specifying abend codes to be percolated by Language Environment

The assembler user exit, when invoked for initialization, might return a list of abend codes that are to be percolated by Language Environment. The list of abend codes is contained in the CEEAUE_A_AB_CODES field of the assembler user exit interface. For more information, see “CEEBXITA assembler user exit interface.”

On non-CICS systems, this list is contained in the CEEAUE_A_AB_CODES field of the assembler user exit interface. (See “CEEBXITA assembler user exit interface.”) Both system abends and user abends can be specified in this list. The abend percolation list specified in the assembler user exit applies to all threads in the enclave.

When TRAP(ON) is in effect, and the abend code is in the CEEAUE_A_AB_CODES list, Language Environment percolates the abend. Normal Language Environment condition handling is never invoked to handle these abends. This feature is useful when you do not want Language Environment condition handling to intervene for certain abends, such as when IMS issues a user abend code 777.

When TRAP(OFF) is specified and there is a program interrupt, the user exit for termination is not driven.

Actions taken for errors that occur within the exit

If any errors occur during the enclave initialization user exit, the standard system action occurs because Language Environment condition handling has not yet been established.

Any errors occurring during the enclave termination user exit lead to abnormal termination (through an abend) of the Language Environment environment.

If there is a program check during the enclave termination user exit and TRAP(ON) is in effect, the application ends abnormally with ABEND code 4044 and reason code 2. If there is a program check during the enclave termination user exit and TRAP(OFF) was specified, the application ends abnormally without additional error checking support. Language Environment performs no condition handling; error handling is performed by the operating system.

Language Environment takes the same actions as described previously for program checks during the process termination user exit.

CEEBXITA assembler user exit interface

You can modify CEEBXITA to perform any function you need, but the exit must have the following attributes after you modify it at installation:

- The user-supplied exit must be named CEEBXITA.
- The exit must be reentrant.
- The exit must be able to execute in AMODE(ANY) and RMODE(ANY).
- The installation-wide guidelines are as follows:

- You must bind (link) the exit with the appropriate Language Environment initialization/termination routines after modification.
- Use the sample customization jobs CEEWDXIT and CEEWCXIT to assist with creating and binding (linking) your exit with Language Environment initialization/termination routines.
- The application-specific guidelines are as follows:
 - You must bind (link) the exit with your application.
 - Use the sample customization job CEEWUXIT and CEEWCXIT to assist with creating your exit.

If a user exit is modified, you are responsible for conforming to the interface shown in Figure 8.

Rule: The modified user exit must be written in assembler.

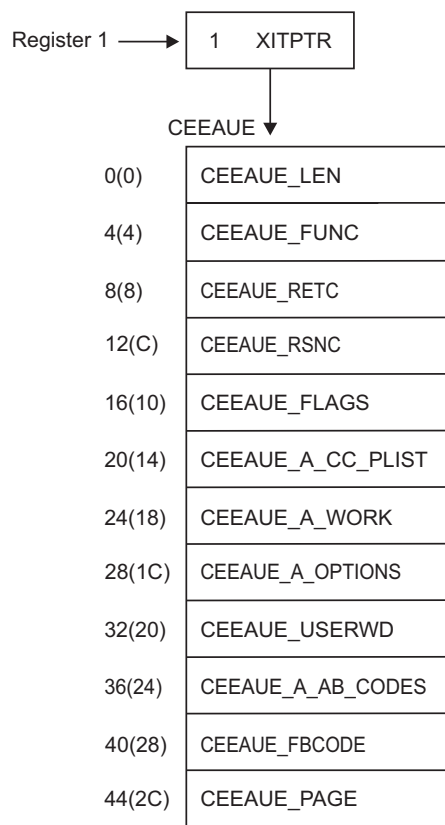


Figure 8. Interface for CEEBXITA assembler user exit

When the user exit is called, register 1 points to a word that contains the address of the CEEAUE control block. The high-order bit is on.

The CEEAUE control block contains the following fullwords:

CEEAUE_LEN (input parameter)

A fullword integer that specifies the total length of this control block. For Language Environment, the length is 48 bytes.

CEEAUE_FUNC (input parameter)

A fullword integer that specifies the function code. Language Environment supports the following function codes:

- 1 Initialization of the first enclave within a process.

- 2 Termination of the first enclave within a process.
- 3 Nested enclave initialization.
- 4 Nested enclave termination.
- 5 Process termination.

The user exit should ignore function codes other than those numbered from 1 through 5.

CEEAEU_RETIC (input/output parameter)

A fullword integer that specifies the return or abend code. CEEAEU_RETIC has different meanings, depending on CEEAEU_ABND:

- If the flag CEEAEU_ABND is off, this fullword is interpreted as the Language Environment return code placed in register 15.
- If the flag CEEAEU_ABND is on, CEEAEU_RETIC is interpreted as an abend code used when an abend is issued. (This could be either an EXEC CICS ABEND or an SVC13.)

CEEAEU_RSNC (input/output parameter)

A fullword integer that specifies the reason code for CEEAEU_RETIC:

- If the flag CEEAEU_ABND is off, this word is interpreted as the Language Environment reason code placed in register 0.
- If the flag CEEAEU_ABND is on, CEEAEU_RETIC is interpreted as an abend reason code used when an abend is issued.

This field is ignored when an EXEC CICS ABEND is issued.

CEEAEU_FLAGS

Contains four 1-byte flags. CEEBXITA uses only the first byte but reserves the remaining flags. All unspecified bits and bytes must be 0. The layout of these flags is shown as follows:

```

Byte 0
    x... .. CEEAEU_ABTERM
    0... .. Normal termination
    1... .. Abnormal termination
    .x.. .. CEEAEU_ABND
    .0.. .. Terminate with CEEAEU_RETIC
    .1.. .. ABEND with CEEAEU_RETIC and CEEAEU_RSNC given
    ..x. .. CEEAEU_DUMP
    ..0. .. If CEEAEU_ABND=0, ABEND with no dump
    ..1. .. If CEEAEU_ABND=1, ABEND with a dump
    ...x .. CEEAEU_STEPS
    ...0 .. ABEND the task
    ...1 .. ABEND the step
    .... 0000 Reserved (must be zero)

Byte 1
    0000 0000 Reserved for future use

Byte 2
    0000 0000 Reserved for future use

Byte 3
    0000 0000 Reserved for future use

```

Byte 0 (CEEAEU_FLAG1) has the following meaning:

CEEAEU_ABTERM (input parameter)

- OFF** Indicates that the enclave is terminating normally (severity 0 or 1 condition).
- ON** Indicates that the enclave is terminating with an Language

Environment return code modifier of 2 or greater. This could, for example, indicate that a severity 2 or greater condition was raised but not handled.

CEEAEU_ABND (input/output parameter)

- OFF** Indicates that the enclave should terminate without an abend being issued. Thus, CEEAEU_RETC and CEEAEU_RSNC are placed into register 15 and register 0 and returned to the enclave creator.
- ON** Indicates that the enclave terminates with an abend. Thus, CEEAEU_RETC and CEEAEU_RSNC are used by Language Environment in the invocation of the abend. During running in CICS, an EXEC CICS ABEND command is issued.

The TRAP runtime option does not affect the setting of CEEAEU_ABND.

When the ABTERMENC(ABEND) runtime option is specified, the enclave always terminates with an abend when there is an unhandled condition of severity 2 or greater, regardless of the setting of the CEEAEU_ABND flag.

CEEAEU_DUMP (output parameter)

- OFF** Indicates that when you request an abend, an abend is issued without requesting a dump.
- ON** Indicates that when you request an abend, an abend requesting a dump is issued.

CEEAEU_STEPS (output parameter)

- OFF** Indicates that when you request an abend, an abend is issued to stop the entire TASK.
- ON** Indicates that when you request an abend, an abend is issued to stop the STEP.

This parameter is applicable only to z/OS; it is ignored under CICS.

CEEAEU_A_CC_PLIST (input/output parameter)

A fullword pointer to the parameter address list of the application program.

If the parameter is not a character string, CEEAEU_A_CC_PLIST contains the register 1 value as passed by the calling program or operating system at the time of program entry.

If the parameter inbound to the MAIN routine is a character string, CEEAEU_A_CC_PLIST contains the address of a fullword address that points to a halfword prefixed string. If this string is altered by the user exit, the string must not be extended in place.

CEEAEU_A_WORK(input parameter)

A fullword pointer to a 256-byte work area that the exit can use. On entry it contains binary zeros and is doubleword-aligned.

This area does not persist across exits.

CEEAEU_A_OPTIONS (output parameter)

Upon return, this field contains a fullword pointer to the address of a halfword-length prefixed character string that contains runtime options. These options are honored only during the initialization of an enclave. When invoked for enclave termination, this field is ignored.

These runtime options override all other sources of runtime options except those that are specified as NONOVR.

Under CICS, the STACK runtime option cannot be modified with the assembler user exit.

CEEAEUE_USERWD (input/output parameter)

A fullword whose value is maintained without alteration and passed to every user exit. Upon entry to the enclave initialization user exit, it is zero. Thereafter, the value of the user word is not altered by Language Environment or any member libraries. The user exit might change the value of this field, and Language Environment maintains that value. This allows the user exit to acquire a work area, initialize it, and pass it to subsequent user exits. The work area might be freed by the termination user exit.

CEEAEUE_A_AB_CODES (output parameter)

During the initialization exit, this field contains a fullword address of a table of abend codes that the Language Environment condition handler percolates while in the (E)STAE exit. Therefore, the application does not have the chance to address the abend. This table is honored prior to shunt routines. The table consists of:

- A fullword count of the number of abend codes that are to be percolated
- A fullword for each of the particular abend codes that are to be percolated

The abend codes might be either user abend codes or system abend codes. User abend codes are specified by F'uuu'. For example, if you want to percolate user ABEND 777, a F'777' would be coded. System abend codes are specified by X'00sss000'.

This parameter is not enabled under CICS.

CEEAEUE_FBCODE (input parameter)

Contains a fullword address of the condition token with which the enclave terminated. If the enclave terminates normally (that is, not due to a condition), the condition token is zero.

CEEAEUE_PAGE (input parameter)

This parameter indicates whether PL/I BASED variables that are allocated storage outside of AREAs are allocated on a 4K-page boundary. You can specify in the field the minimum number of bytes of storage that must be allocated. Your allocation request must be an exact multiple of 4K.

The IBM-supplied default setting for CEEAEUE_PAGE is 32768 (32K).

If CEEAEUE_PAGE is set to zero, PL/I BASED variables can be placed on other than 4K-page boundaries.

CEEAEUE_PAGE is honored only during enclave initialization, that is, when CEEAEUE_FUNC is 1 or 3.

The offset of CEEAEUE_PAGE under Language Environment is different than under OS PL/I Version 2 Release 3.

Parameter values in the assembler user exit

The parameters described in “CEEBXITA assembler user exit interface” on page 186 contain different values depending on how the user exit is used. Table 21 on page 191 and Table 22 on page 192 describe the possible values for the parameters based on how the assembler user exit is invoked.

Table 21. Parameter values in the assembler user exit (Part 1). The assembler user exit contains these parameter values depending on when it is invoked.

When invoked	CEEAEU_ LEN	CEEAEU_RETC	CEEAEU_RSNC	CEEAEU_FLAGS	CEEAEU_A_CC_PLIST
First Enclave within Process Initialization — Entry CEEAEU_FUNC = 1	48	0	0	0	Upon entry, CEEAEU_A_CC_PLIST contains the register 1 value from the operating system. It can contain both runtime options and user parameters. You can alter it in a user exit. Upon return, the CEEAEU_A_CC_PLIST is processed and merged as the invocation string.
First Enclave within Process Initialization — Return		0, or abend code if CEEAEU_ABND = 1	0, or reason code for CEEAEU_RETC if CEEAEU_ABND = 1	See Note 1 on page 192.	Register 1, used as the new parameter list. CEEAEU_A_CC_PLIST can contain both runtime options and user parameters. You can alter it in a user exit. Upon return, the CEEAEU_A_CC_PLIST is processed and merged as the invocation string.
First Enclave within Process Termination — Entry CEEAEU_FUNC = 2	48	Return code issued by application that is terminating.	Reason code that accompanies CEEAEU_RETC.	See Note 2 on page 192.	
First Enclave within Process Termination — Return		If CEEAEU_ABND = 0, the return code placed into register 15 when the enclave terminates. If CEEAEU_ABND = 1, the abend code.	If CEEAEU_ABND = 0, the enclave reason code. If CEEAEU_ABND = 1, the abend reason code.	See Note 1 on page 192.	
Nested Enclave Initialization — Entry CEEAEU_FUNC = 3	48	0	0	0	The register 1 value discovered in a nested enclave creation. CEEAEU_A_CC_PLIST can contain both runtime options and user parameters. You can alter it in a user exit. Upon return, the CEEAEU_A_CC_PLIST is processed and merged as the invocation string.
Nested Enclave Initialization — Return		0, or if CEEAEU_ABND = 1, the abend code.	0, or if CEEAEU_ABND = 1, reason code for CEEAEU_RETC.	See Note 1 on page 192.	Register 1 used as the new enclave parameter list. CEEAEU_A_CC_PLIST can contain both runtime options and user parameters. You can alter it in a user exit. Upon return, the CEEAEU_A_CC_PLIST is processed and merged as the invocation string.
Nested Enclave Termination — Entry CEEAEU_FUNC = 4	48	Return code issued by enclave that is terminating.	Reason code accompanying CEEAEU_RETC.	See Note 2 on page 192.	
Nested Enclave Termination — Return		If CEEAEU_ABND = 0, the return code from the enclave. If CEEAEU_ABND = 1, the abend code.	If CEEAEU_ABND = 0, the enclave reason code. If CEEAEU_ABND = 1, the enclave reason code.	See Note 1 on page 192.	

Table 21. Parameter values in the assembler user exit (Part 1) (continued). The assembler user exit contains these parameter values depending on when it is invoked.

When invoked	CEEAEU_ LEN	CEEAEU_RETC	CEEAEU_RSNC	CEEAEU_FLAGS	CEEAEU_A_CC_PLIST
Process Termination — Entry Function Code = 5	48	Return code presented to the invoking system in register 15 that reflects the value returned from the “first enclave within process termination”.	Reason code accompanying CEEAEU_RETC that is presented to the invoking system in register 0 and reflects the value returned from the “first enclave within process termination”.	See Note 3.	
Process Termination — Return		If CEEAEU_ABND = 0, return code from the process. If CEEAEU_ABND = 1, the abend code.	If CEEAEU_ABND = 0, the reason code for CEEAEU_RETC from the process. If CEEAEU_ABND = 1, reason code for the CEEAEU_RETC abend reason code.	See Note 1.	

Notes:

1. CEEAEU_FLAGS:

CEEAEU_ABND = 1 if an abend is requested, or 0 if the enclave should continue with termination processing
CEEAEU_DUMP = 1 if the abend should request a dump
CEEAEU_STEPS = 1 if the abend should abend the step
CEEAEU_STEPS = 0 if the abend should abend the task

2. CEEAEU_FLAGS:

CEEAEU_ABTERM = 1 if the application is terminating with an Language Environment return code modifier of 2 or greater, or 0 otherwise
CEEAEU_ABND = 1 if an abend is requested, or 0 if the enclave should continue with termination processing
CEEAEU_DUMP = 0
CEEAEU_STEPS = 0

3. CEEAEU_FLAGS:

CEEAEU_ABTERM = 1 if the last enclave is terminating abnormally (that is, a Language Environment return code modifier is 2 or greater). This reflects the value returned from the “first enclave within process termination”.
CEEAEU_ABND = 1 if an abend is requested, or 0 if the enclave should continue with termination processing “first enclave within process termination” (function code 2).
CEEAEU_DUMP = 0
CEEAEU_STEPS = 0

Table 22. Parameter values in the assembler user exit (Part 2). The assembler user exit contains these parameter values depending on when it is invoked.

When invoked	CEEAEU_A_WORK	CEEAEU_A_OPTIONS	CEEAEU_USERWD	CEEAEU_A_AB_CODES	CEEAEU_FBCODE	CEEAEU_PAGE
First Enclave within Process Initialization — Entry CEEAEU_FUNC = 1	Address of a 256-byte work area of binary zeros.		0		0	Minimum number of storage bytes to be allocated for PL/I BASED variables (default = 32768).
First Enclave within Process Initialization — Return		Pointer to address of a halfword prefixed character string containing runtime options, or 0.	Value of CEEAEU_USERWD for all subsequent exits.	Pointer to the abend codes table, or 0.		User specified PAGE value. Minimum number of storage bytes to be allocated for PL/I BASED variables (default = 32768).
First Enclave within Process Termination — Entry CEEAEU_FUNC = 2	Address of a 256-byte area of binary zeros.		Return value from previous exit.		Feedback code causing termination.	
First Enclave within Process Termination — Return			The value of CEEAEU_USERWD for all subsequent exits.			
Nested Enclave Initialization — Entry CEEAEU_FUNC = 3	Address of a 256-byte work area of binary zeros.		Return value from previous exit.		0	Minimum number of storage bytes to be allocated for PL/I BASED variables (default = 32768).

Table 22. Parameter values in the assembler user exit (Part 2) (continued). The assembler user exit contains these parameter values depending on when it is invoked.

When invoked	CEEAEU_A_WORK	CEEAEU_A_OPTIONS	CEEAEU_USERWD	CEEAEU_A_AB_CODES	CEEAEU_FBCODE	CEEAEU_PAGE
Nested Enclave Initialization — Return		Pointer to fullword address that points to a halfword prefixed length string containing runtime options, or 0.	The value of CEEAEU_USERWD for all subsequent exits.	Pointer toabend codes table, or 0.		User specified PAGE value. Minimum number of storage bytes to be allocated for PL/I BASED variables (default = 32768).
Nested Enclave Termination — Entry	Address of a 256-byte work area of binary zeros.		Return value from previous exit.		Feedback code causing termination.	
CEEAEU_FUNC = 4						
Nested Enclave Termination — Return			Value of CEEAEU_USERWD for all subsequent exits.			
Process Termination — Entry CEEAEU_FUNC = 5	Address of a 256-byte work area of binary zeros.		Return value from previous exit.		Feedback code causing termination.	
Process Termination — Return			Value of CEEAEU_USERWD for all subsequent exits.			

Abnormal termination exit

The abnormal termination exits in CEEEXTAN are invoked during the termination of an enclave due to an unhandled condition of severity 2 or greater. An abnormal termination exit is invoked in AMODE(31), with register 12 pointing to the CAA and register 13 pointing to a DSA with a valid NAB.

For AMODE 64 applications, an abnormal termination exit is invoked in AMODE(64), with register 4 pointing to the caller's DSA and register 1 pointing to the CIB.

For more information about creating and using abnormal termination exits, see “CEEEXTAN abnormal termination exit CSECT” on page 150.

Syntax

Abnormal_Termination_Exit (CIBPTR)

CIBPTR (INPUT)

A pointer to the condition information block for the current condition.

Usage notes for AMODE 31 applications

- The abnormal termination exit must be written in assembler. If you write an abnormal termination exit in Language Environment-enabled assembler, be sure to specify MAIN=NO in the CEENTRY macro.
- The abnormal termination exit cannot call any HLL programs.
- The abnormal termination exit cannot create a Language Environment enclave.
- The abnormal termination exit can use the following Language Environment callable services if the feedback code is passed as a parameter:
 - Date and time callable services
 - Dynamic storage callable services
 - Message handling callable services
 - National language support callable services

- A subset of the general callable services: CEE3DMP, CEE3GRC, CEE3PRM
- A subset of the condition handling callable services: CEE3GRN, CEEDCOD, CEEGPID, CEEGQDT, CEEITOK, CEENCOD

In addition, observe the restrictions on the use of system services as described in *z/OS Language Environment Programming Guide*.

- Language Environment issues a system-dependent LOAD for one of the names contained in CEEEXTAN. If the load is successful, the abnormal termination exit is invoked.
- Upon return from the abnormal termination exit, Language Environment deletes the routine. A return code is not provided, because Language Environment takes no action (beyond deleting the routine) for a nonzero return code.
- If Language Environment intercepts a program check, an ABEND, or a CEESGL while an abnormal termination exit is in control, Language Environment issues an ABEND to terminate the enclave with the abend code 4087 reason code 10.
- Entry conditions into the abnormal termination exit:

Register 1

Has a standard OS parameter list as described above.

Register 12

Points to the CAA.

Register 13

Points to a Language Environment DSA with a valid NAB. (You can use it as a standard 18-fullword save area.)

Register 14

Contains the return address.

Register 15

Contains the entry point address.

AMODE

Is 31.

- Exit conditions from the abnormal termination exit:

Registers 15–1

Undefined.

Registers 2–13

Unchanged.

Register 14

Is the return point.

AMODE

Is 31.

Usage notes for AMODE 64 applications

- See the CEEWQATX sample abnormal termination exit routine for a coding example.
- Specify FETCHABLE=RENT on the CELQPRLG macro.
- The abnormal termination exit should call only the following C library functions:
 - Date and time functions
 - Dynamic storage functions
 - Message handling functions
 - National language support functions
 - Dump-oriented and CIB-oriented functions

- printf() and related functions
- Language Environment issues fetch() to load each routine named in CELQXTAN. If the fetch is successful, the routine is called. Any return code from the called routine is ignored, and release() is called to delete the routine.
- If Language Environment intercepts a program check or ABEND while an abnormal termination exit is in control, ABEND 4087 with reason code 10 is issued to end the application.
- Registers at entry to the AMODE 64 abnormal termination exit:

Register 1

Points to CIB

Register 4

Caller's DSA pointer (biased)

Register 5

Pointer to WSA (if available)

Register 6

Address of called entry point

Register 7

Return address

The other register contents are unspecified.

Load notification user exit

The purpose of the load notification user exit is to provide customers running applications with LRR active the ability to improve performance by preventing the use count for frequently used modules from dropping below one.

The load notification user exit registered via CEEBLNUE is invoked:

- Once during region initialization processing
- After each successful load of a module by Language Environment
- Once during region termination processing.

When invoked during region initialization processing, the load notification user exit can initialize some form of module list which can be used during subsequent invocations to keep track of modules which have an extra load.

When invoked after the successful load of a module, the load notification user exit would perform an extra load, if desired, and update the module list.

When invoked during region termination processing, the load notification user exit would delete the modules identified in the list so that they are actually removed from storage.

The load notification user exit will be loaded and called only when Library Routine Retention (LRR) is active.

For more information about creating and using the load notification user exit, see Chapter 7, "Customizing user exits," on page 145.

Syntax: Load_Notification_User_Exit (CEELNUEPTR)

CEELNUEPTR (INPUT)

A pointer to the load notification user exit control block.

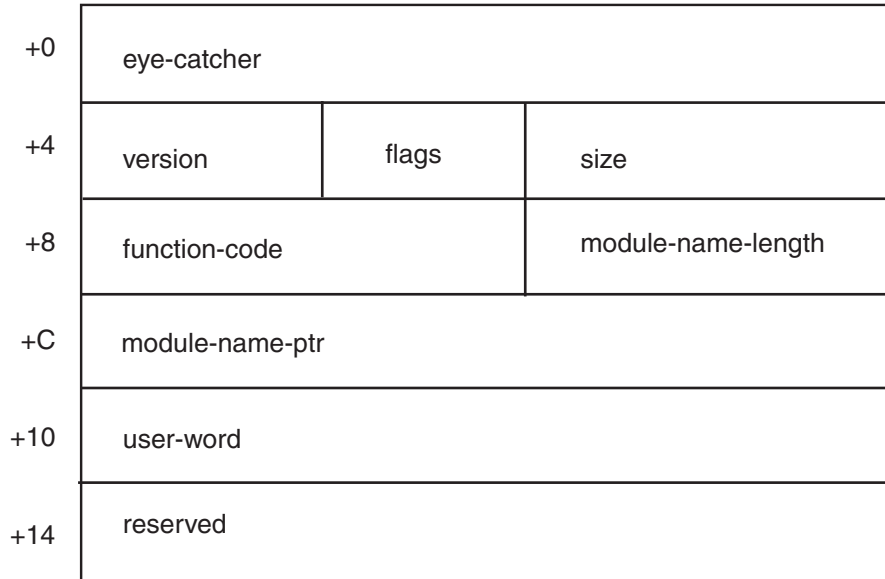


Figure 9. CEELNUE control block map

The CEELNUE control block elements shown in Figure 9 are described as follows:

eye-catcher **(INPUT)**

A 4-byte character field containing "LNUE" indicating this is the CEELNUE control block.

version **(INPUT)**

A 1-byte binary field containing the control block version. This field is set to 0x01 for the first version.

flags **(INPUT)**

A 1-byte binary field containing flags as shown:

```
x... .... load-type
0... .... OS
1... .... HFS
.000 0000 reserved
```

The flags are defined as follows:

load-type **(INPUT)**

```
0      OS module was loaded
1      z/OS UNIX module was loaded
```

size **(INPUT)**

A 2-byte integer field containing the size of the CEELNUE control block. This field will be set to 0x0018 for the first version.

function-code **(INPUT)**

A 2-byte integer field containing the code of the function being performed when the load notification user exit gets control. The following function code values for the first version are:

```
0      Initialization
1      Load
2      Termination
```

module-name-length **(INPUT)**

A 2-byte integer field. When the function-code is 1, this field contains the

length of the name of the module that was just loaded. This value will be 8 when an OS module was loaded. When the function-code is 0 or 2 the value should be ignored.

module-name-ptr **(INPUT)**

A 4-byte address. When the function-code is 1, this field contains the address of the name of the module that was just loaded. The module name pointed to will be 8 characters in length, padded on the right with blanks as necessary, when an OS module was loaded. When the function-code is 0 or 2 the value should be ignored.

user-word **(INPUT/OUTPUT)**

A 4-byte binary field. This field is to be used to communicate information between successive calls to the load notification user exit. The very first time the load notification user exit is called, this field will be 0. The load notification user exit may modify the value in this field. The value will be saved by Language Environment and returned on subsequent invocations. Language Environment will only honor changing the field during the initialization function.

reserved

A 4-byte reserved field.

Usage notes

- The load notification user exit must be written in Assembler. If you write the load notification user exit in Language Environment-enabled assembler, be sure to specify MAIN=NO on the CEEENTRY macro.
- The load notification user exit must not call any HLL programs.
- The load notification user exit must not create a Language Environment enclave.
- The load notification user exit must not use any Language Environment services.
- Language Environment issues a system dependent LOAD for the name contained in CEEBLNUE. If the load is successful, then the load notification user exit is invoked.
- Upon return from the load notification user exit, Language Environment takes no action other than continuing with its processing, as no return codes are defined.
- The load notification user exit is invoked in AMODE(31). The RMODE can be either ANY or 24.
- Registers provided on entry to the load notification user exit are:

Register 1

Points to a word which contains the address of the CEELNUE control block.

Register 12

Points to the CAA.

Register 13

Points to a standard save area. The exit routine can save the registers here across its processing.

Register 14

Contains the return address.

Register 15

Contains the entry point address upon entry.

- Registers provided on exit to the load notification user exit are:

Registers 15-1

Undefined.

Registers 2-13

Unchanged.

Register 14

Contains the return address.

Storage tuning user exit

When Language Environment is used in transaction processing environments where Language Environment enclaves are constantly being initialized and terminated, such as CICS online and IMS/TM message processing regions, tuning the Language Environment storage options can improve the performance of the transactions. By tuning the Language Environment storage options, it can reduce the time spent doing system GETMAINS and it can reduce the time spent initializing Language Environment stack and heap increments.

The Language Environment storage tuning user exit is provided to help manage the task of setting Language Environment storage option values that provide the best performance in transaction processing environments where Language Environment enclaves are constantly being initialized and terminated. The Language Environment storage tuning user exit can be used to manage the Language Environment storage values for your main programs without having to statically link-edit the storage values with your load modules.

Note: When running on CICS, the Language Environment automatic storage tuning for CICS may provide the storage tuning function you need without having to use the Language Environment storage tuning user exit. See “Language Environment automatic storage tuning for CICS” on page 167 for a discussion of Automatic Storage Tuning (AUTOTUNE) for CICS.

The Language Environment storage tuning user exit provides a programming interface that allows you to:

- Collect Language Environment storage tuning information without having to run with the RPTSTG option.
- Set the Language Environment runtime options STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP for each Language Environment enclave.

Note: Vendor Heap Manager (VHM) is not handled by the Language Environment storage tuning user exit.

- Alter the behavior of Language Environment automatic storage tuning for CICS.

The storage tuning exit can be used when running on CICS or when running on non-CICS with library routine retention. The name of the exit is as follows:

- CEECSTX for CICS
- CEEBSTX for non-CICS

The storage tuning exit is loaded when Language Environment loads its other runtime load modules. On CICS, the storage tuning exit is loaded during CICS startup. On non-CICS with LRR, the storage tuning exit is loaded when Language Environment is initialized to run the first program. If the load of the storage tuning exit is not successful, Language Environment does not issue a message. It also does not attempt to call the storage tuning exit.

The storage tuning user exit is called at certain times during Language Environment processing.

- Region initialization
- Region termination
- Enclave initialization
- Enclave termination
- New load module (CICS only)

Region initialization

The storage tuning user exit is called when Language Environment is initializing its region level resources. On CICS, this call occurs only once during CICS startup. On non-CICS, this call only occurs once when Language Environment is initialized to run the first program. At this point the storage tuning user exit is passed a user word. The storage tuning user exit can acquire any resources it needs. For example it may allocate memory for an incore table and put the address of the table in the user word.

Region termination

The storage tuning user exit is called when Language Environment is terminating its region level resources. At this point the storage tuning user exit is passed a user word. The storage tuning user exit can free any resources it had obtained.

Enclave initialization

The storage tuning user exit is called after the assembler user exit is called for enclave initialization. At this point the storage tuning user exit is passed information about the main program, a user word, and an area to provide Language Environment storage option values.

When running on non-CICS or on CICS without automatic storage tuning, the storage tuning user exit can do one or both of the following:

- Provide Language Environment storage option values. The storage option values for each program that needs to be tuned could be in a file or in an incore table. The address of an incore table could be stored in the user word. The storage tuning user exit could look up the storage values for the program and pass them back to Language Environment.
- Request that Language Environment collect storage tuning information. By requesting storage tuning information, it will cause Language Environment to collect storage tuning information and then call the storage tuning user exit when the enclave terminates.

When running on CICS with automatic storage tuning, the storage tuning user exit has limited function. At the enclave initialization call, any storage option values provided are ignored. If the storage tuning user exit wants to provide storage option values, it has to provide them at the new load module call and at the enclave termination call. The storage tuning user exit can turn off the request by automatic storage tuning to collect storage allocation information.

Enclave termination

The storage tuning user exit is called after the assembler user exit is called for enclave termination. At this point the storage tuning user exit is passed information about the main program, a user word, and the storage tuning information.

When running on non-CICS or on CICS without automatic storage tuning, the storage tuning user exit can do the following:

- Take the storage tuning information for the program that you want to tune, and put the information in a file or in an incore table. The tuning information can be saved and used in the enclave initialization call the next time the program is used. Or the tuning information may be written to a file and processed at a later time to determine the best tuning values.

When running on CICS with automatic storage tuning, the storage tuning user exit can do the following:

- Provide storage options values to override the values set by Language Environment automatic storage tuning.

Note: At enclave initialization, if the storage tuning user exit does not request Language Environment to collect storage tuning information, then Language Environment will not call the storage tuning user exit at enclave termination.

New load module (CICS only)

At this point the exit is passed information about the main program, a user word, and a control block to provide storage option values.

The storage tuning user exit is called whenever CICS asks Language Environment for the size of the preallocated storage area to be used for Language Environment stacks and heaps. CICS makes this call whenever it is processing the first copy or a new copy of a load module

When running on CICS without automatic storage tuning, the storage tuning user exit can do the following:

1. It can enable or disable storage tuning for a module.
2. If the load module is enabled for the storage tuning, it can provide the storage values. This information is then used by Language Environment to tell CICS to update its information about how much preallocated storage should be allocated for the program before CICS calls Language Environment to run the program.

The storage tuning values passed back from the storage tuning user exit for this call should match the values that the storage tuning user exit passes back in the enclave initialization call. If this is not done, preallocated storage allocations performed by CICS for Language Environment will not be the optimal size, and may result in additional GETMAINS.

When running on CICS with automatic storage tuning, the storage tuning user exit can do the following:

1. It can enable or disable Language Environment storage tuning for the load module.
2. If automatic storage tuning is wanted for the load module, the storage tuning user exit can provide the starting storage values used by Language Environment automatic storage tuning.

Using the storage tuning user exit

The storage tuning user exit can be used on non-CICS or on CICS without automatic storage tuning to:

- Collect Language Environment storage tuning information without having to run with the RPTSTG option.
- Set the Language Environment runtime storage options for each Language Environment enclave.

A sample storage tuning user exit for CICS called CEEWCSTX is available in SCEESAMP. A sample storage tuning user exit for non-CICS called CEEWBSTX is available in SCEESAMP.

Using the storage tuning user exit to collect information

If you want to use the storage tuning user exit to collect storage usage information instead of using RPTSTG(ON), you can use a storage tuning user exit that always requests that Language Environment collect storage tuning information.

When running on non-CICS, the behavior of the storage tuning user exit could be as follows:

- At the region initialization call, the storage tuning user exit opens a file for output. The storage tuning user exit will write the Language Environment storage information to the file at enclave termination.
- At the enclave initialization call, the storage tuning user exit always requests Language Environment to collect storage information.
- At the enclave termination call, the storage tuning user exit can make a call to the system to determine the load module name of the main program by using CSVQUERY, and then it can write a record to a file which includes the load module name and the storage usage information. On the CSVQUERY call, you use the address of the main program that is passed to the storage tuning user exit as input in the INADDR parameter, and use the OUTEPNM parameter to get the load module name.
- At the region termination call, the storage tuning user exit closes the file it opened.

When running on CICS, the behavior of the storage tuning user exit could be as follows:

- At the region initialization call, the storage tuning user exit returns immediately.
- At the new load module call, the storage tuning user exit always enables the use of the storage tuning user exit for the load module.
- At the enclave initialization call, the storage tuning user exit always requests Language Environment to collect storage information.
- At the enclave termination call, the storage tuning user exit gets the name of the module, and then writes a record to a file where the record includes the load module name and the storage usage information.
- At the region termination call, the storage tuning user exit returns immediately.

After the file is closed, the data in the file can be analyzed. Once the data has been analyzed, you might decide to alter your runtime option defaults, or provide a CEEUOPT to certain modules, or use the storage tuning user exit to provide storage values.

Using the storage tuning user exit to provide storage values

If you want to use the storage tuning user exit to perform storage tuning of certain load modules that start Language Environment enclaves you can use a storage

tuning user exit that provides values for the Language Environment runtime options STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP.

When running on non-CICS, the behavior of the storage tuning user exit could be as follows:

- At the region initialization call, the storage tuning user exit allocates and initializes a table that has storage tuning information for the load modules it wants to tune.

The storage tuning user exit would need to know which load modules it wants to tune and it would need to find the storage values it wants to use for each load module. These values could be in the storage tuning user exit itself as constants, or could be in a file.

- At the enclave initialization call, the storage tuning user exit would do a lookup in the table to see if the load module has storage tuning information. If it does, the storage tuning information is provided back to Language Environment.

Note: If the modules to be tuned are preloaded at region initialization and their entry point addresses are kept in the table, a CSVQUERY would not need to be done every time the module is run. Instead, a search of the table could be done using the entry point address.

- At the region termination call, the storage tuning user exit frees any resources it acquired.

Using the storage tuning user exit to provide storage values

When running on CICS, the behavior of the storage tuning user exit could be as follows:

- At the region initialization call, the storage tuning user exit allocates and initializes a table that has storage tuning information for the load modules it wants to tune.

The storage tuning user exit would need to know which load modules it wants to tune and it would need to find the storage values it wants to use for each load module. These values could be in the storage tuning user exit itself as constants, or could be in a file.

- At the new load module call, the storage tuning user exit would do a lookup in the table to see if the load module has storage tuning information. If it does, the load module is enabled for storage tuning and the storage tuning information is provided back to Language Environment.
- At the enclave initialization call, the storage tuning user exit would do a lookup in the table for the storage tuning information and the storage tuning information is provided back to Language Environment.
- At the region termination call, the storage tuning user exit frees any resources it acquired.

Storage tuning user exit interface

The storage tuning user exit can be used when running on CICS or when running on non-CICS with library routine retention. The name of the exit is as follows:

- CEECSTX for CICS
- CEEBSTX for non-CICS

The storage tuning exit is loaded when Language Environment loads its other runtime load modules. If the load of the storage tuning exit is not successful, Language Environment will not issue a message and will not attempt to call the storage tuning exit.

The storage tuning user exit must be written in assembler and must be reentrant.

Syntax: Storage_Tuning_User_Exit (CEESTXPTR)

CEESTXPTR (INPUT)

A pointer to the storage tuning user exit control block (CEESTX). Figure 10 shows a mapping of the storage tuning user exit control block.

+0	version	flags	reserved
+4	function-code		
+8	user-word		
+C	program-entry-point		
+10	addr-CEEUOPT		
+14	addr-CICS-info		
+18	addr-storage-info		

Figure 10. CEESTX control block map

The CEESTX elements are described as follows:

version (INPUT)

A 1-byte field containing the control block version. This field will be set to 0x02 since this is the second version.

flags (INPUT/OUTPUT)

A 1-byte field containing flags. The layout of these flags is as follows:

```
x... .... collect-storage-usage
.x... .... collect-storage-alloc
..00 0000 reserved
```

A one-byte field containing flags. The flags are defined as follows:

collect-storage-usage (INPUT/OUTPUT)

This flag is used to tell Language Environment to collect storage usage information. Language Environment only uses this flag when the storage tuning user exit is called during enclave initialization and automatic storage tuning for CICS is not running.

0 Do not collect storage usage information.

1 Collect storage usage information and call the storage tuning user exit during enclave termination with the storage usage information.

Note: When the storage tuning user exit is used to collect storage usage information, it will increase the time it takes for an application to run.

collect-storage-alloc (INPUT/OUTPUT)

This flag is used to tell Language Environment to collect storage allocation information. Language Environment only uses this flag when the storage tuning user exit is called during enclave initialization. The flag values are:

- 0 Do not collect storage allocation information.
- 1 Collect storage allocation information and call the storage tuning user exit during enclave termination with the storage allocation information.

Note: There is significantly less overhead collecting storage allocation information compared to collecting storage usage information.

function-code (INPUT)

A 4-byte integer field containing the code of the function being performed when the storage tuning user exit gets control. The function code values are:

- 1 Region initialization
- 2 Region termination
- 3 Enclave initialization
- 4 Enclave termination
- 5 New load module (CICS only)

user-word (INPUT/OUTPUT)

A fullword that can be used to communicate information between successive calls to the storage tuning user exit. The first time the storage tuning user exit is called, this field is 0. The storage tuning user exit may modify the value in this field when it is called for region initialization. The value is saved by Language Environment and returned on subsequent invocations.

program-entry-point (INPUT)

When the storage tuning user exit is called for region initialization and region termination, this field is zero.

When the storage tuning user exit is called for enclave initialization, enclave termination, and new load module, this field contains the entry point address of the main program.

Note: When C, C++, or PL/I is the main program, this field contains the address of the main program and not the address of CEESTART.

addr-CEEUOPT (INPUT)

When the storage tuning user exit is called for region initialization and region termination, this field is zero.

When the storage tuning user exit is called for enclave initialization, enclave termination, and new load module, this field contains the address of the CEEUOPT link-edited with the main program. If no CEEUOPT is link-edited with the main program, this field is zero.

addr-CICS-info (INPUT)

When running on non-CICS, this field is set to zero.

When running on CICS, this field contains the address of the CEESTX CICS specific control block.

See Figure 11 on page 205 for a mapping of the control block.

+0	flags	reserved
+4	addr-SYSEIB	
+8	addr-load-module-name	
+C	addr-autotune-storage-settings	
+10	addr-autotune-storage-override	

Figure 11. CEESTX CICS-specific control block map

The CEESTX CICS-specific control block elements are described as follows:

flags (**INPUT/OUTPUT**)

When the storage tuning user exit is called for region initialization and region termination, this field is reserved.

When the storage tuning user exit is called for enclave initialization, enclave termination, and new load module, this field contains a 1-byte area containing flags. The layout of these flags is as follows:

```
x... .... load-mod-eligible
.x... .... automatic tuning
.000 0000 reserved
```

The flags are defined as follows:

load-mod-eligible (**INPUT/OUTPUT**)

For each load module loaded by CICS, there is a unique load-mod-eligible flag available to the storage tuning user exit. The flag is input/output when the storage tuning user exit is called for the new load module function. The flag is output only when the storage tuning user exit is called for enclave initialization and enclave termination.

When Language Environment automatic storage tuning for CICS is not being used, the initial value of the flag is zero. When Language Environment automatic storage tuning for CICS is being used, the initial value of the flag indicates if Language Environment automatic storage tuning for CICS will be performing automatic storage tuning for enclaves started to run the load module.

This flag is used by the storage tuning user exit to indicate to Language Environment if the storage tuning user exit should be called for enclave initialization when the load module is called to start an enclave. For example, when the storage tuning user exit is called for the new load module function, it can determine if it wants to tune the storage options for the enclaves that are started to run the load module. If the storage tuning user exit decides it does want to tune the enclaves for this load module, it must set the flag on.

- 0 Do not call the storage tuning user exit when the program is used to start an enclave. If Language Environment automatic storage tuning for CICS is being used, do not perform automatic storage tuning when the program is used to start an enclave.
- 1 Call the storage tuning user exit when the program is used to start a Language Environment enclave. If Language

Environment automatic storage tuning for CICS is being used, perform automatic storage tuning when the program is used to start an enclave.

automatic-tuning **(INPUT)**

This bit indicates if automatic storage tuning for CICS is being used.

0 Automatic storage tuning for CICS is not being used.

1 Automatic storage tuning for CICS is being used.

addr-SYSEIB **(INPUT)**

A pointer to the CICS system EIB (EXEC Interface Block).

addr-load-mod-name **(INPUT)**

When the storage tuning user exit is called for region initialization and region termination, this field is zero.

When the storage tuning user exit is called for enclave initialization, enclave termination, and new load module, this field contains:

- A pointer to an 8 byte character field that has the name of the load module loaded by CICS if you are running with CICS Transaction Server Release 2 or Release 3 with APAR PQ31262 or with CICS/ESA Version 4 with APAR PQ31185.
- A zero if you are not running with the APARs listed above.

Note: The storage tuning user exit will be called for the new load module function for every program that is Language Environment-enabled. Not every program loaded by CICS will be the main program.

addr-autotune-storage-settings **(INPUT)**

When the storage tuning user exit is called for region initialization and region termination, this field is zero. When the storage tuning user exit is called for enclave initialization, enclave termination, and new load module, this field contains:

- A zero when running on a CICS region that is not running with automatic storage tuning for CICS.
- A pointer to a copy of the CEESTX storage values control block when running on a CICS region using automatic storage tuning for CICS. This control block has the storage values that is used by Language Environment for automatic storage tuning. There is a copy of this control block for each load module. This control block is an input only control block and must not be changed by the storage tuning user exit.

addr-autotune-storage-override **(INPUT)**

When the storage tuning user exit is called for region initialization, region termination, enclave initialization, and new load module, this field is zero. When the storage tuning user exit is called for enclave termination this field contains:

- A zero when running on a CICS region that is not running with automatic storage tuning for CICS.
- A pointer to a copy of the CEESTX storage values control block when running on a CICS region using Automatic Storage Tuning for CICS. This control block can be changed by the storage tuning user exit as a way to override the initial size values set by Automatic Storage Tuning for CICS. Language Environment initializes the flags in the first word of the CEESTX storage values control block to hex zeros before calling the storage tuning user exit.

addr-storage-info (**INPUT**)

When the storage tuning user exit is called for region initialization and region termination, this field is zero.

When the storage tuning user exit is called for enclave initialization, enclave termination, and new load module, this field contains the address of a control block that is used to pass Language Environment storage information between Language Environment and the storage tuning user exit. There are three forms of the control block:

- The CEESTX storage values control block
- The CEESTX storage used control block
- The CEESTX storage allocated control block.

When the storage tuning user exit is called with the enclave initialization function or the new load module function, the CEESTX storage values control block is passed.

When running on non-CICS or on CICS without automatic storage tuning:

- All of the fields in the control block are output only fields except for the first word. The first word is an input/output field. Language Environment initializes the flags in the first word to hex zeros before calling the storage tuning user exit.
- All other fields will not be initialized.

When running on CICS with automatic storage tuning:

- All of the fields in the control block are output only fields except for the first word. The first word is an input/output field. Language Environment initializes the flags in the first word to hex zeros before calling the storage tuning user exit.
- All other fields will not be initialized.
- The storage option values provided at the new load module call are used as the starting values by automatic storage tuning.
- The storage option value provided at enclave initialization call are ignored.

See Figure 12 on page 208 for a mapping of the CEESTX storage values control block.

+0	stg-flags	reserved
+4	STACK-flags	reserved
+8	STACK-init-size	
+C	STACK-incr-size	
+10	LIBSTACK-flags	reserved
+14	LIBSTACK-init-size	
+18	LIBSTACK-incr-size	
+1C	HEAP-flags	reserved
+20	HEAP-init-size	
+24	HEAP-incr-size	
+28	HEAP-init-size24	
+2C	HEAP-incr-size24	
+30	ANYHEAP-flags	reserved
+34	ANYHEAP-init-size	
+38	ANYHEAP-incr-size	
+3C	BELOWHEAP-flags	reserved
+40	BELOWHEAP-init-size	
+44	BELOWHEAP-incr-size	

Figure 12. Mapping of the CEESTX storage values control block

When the storage tuning user exit is called with the enclave termination function, the CEESTX storage used control block or the CEESTX storage allocated control block is passed to the storage tuning user exit to provide storage information collected by Language Environment. The CEESTX storage used control block is passed when the storage tuning user exit requested Language Environment to collect storage *usage* information. The CEESTX storage allocated control block is passed when the storage tuning user exit requested Language Environment to collect storage *allocation* information. All of the fields in the control block are input only. See Figure 13 on page 212 for a mapping of the CEESTX storage used control block. See Figure 14 on page 213 for a mapping of the CEESTX storage allocated control block map.

The CEESTX storage values control block elements are described as follows:

stg-flags (OUTPUT)

A 1-byte field containing flags. The layout of these flags is as follows:

```

0... .... STACK options not provided
1... .... STACK options provided
.0.. .... LIBSTACK options not provided
.1.. .... LIBSTACK options provided
..0. .... HEAP options not provided
..1. .... HEAP options provided
...0 .... ANYHEAP options not provided
...1 .... ANYHEAP options provided
.... 0... BELOWHEAP options not provided
.... 1... BELOWHEAP options provided
.... .000 reserved

```

There is a flag for each storage runtime option that can be altered by the storage tuning user exit. The exit must turn on the flags for the storage runtime options for which it is providing values.

STACK-flags (OUTPUT)

A 2-byte field containing flags. The layout of these flags is as follows:

```

Byte 0
0... .... STACK initial size not provided
1... .... STACK initial size provided
.0.. .... STACK increment size not provided
.1.. .... STACK increment size provided
..0. .... STACK location not provided
..1. .... STACK location provided
...0 .... STACK disposition not provided
...1 .... STACK disposition provided
.... 0000 reserved

Byte 1
0... .... STACK location ANYWHERE
1... .... STACK location BELOW
.0.. .... STACK disposition KEEP
.1.. .... STACK disposition FREE
..00 0000 reserved

```

In byte 0, there is a flag for each STACK suboption. The exit must set the flags in byte 0 to indicate the STACK suboptions for which it is providing values.

The STACK location can be set to ANYWHERE only if ALL31(ON) is in effect. If the STACK location is set to ANYWHERE, and ALL31(OFF) is in effect, the STACK location is not changed.

STACK-init-size (OUTPUT)

A fullword binary field used to indicate the number of bytes for the STACK initial size.

STACK-incr-size (OUTPUT)

A fullword binary field used to indicate the number of bytes for the STACK increment size.

LIBSTACK-flags (OUTPUT)

A 2-byte field containing flags. The layout of these flags is as follows:

```

Byte 0
0... .... LIBSTACK initial size not provided
1... .... LIBSTACK initial size provided
.0.. .... LIBSTACK increment size not provided
.1.. .... LIBSTACK increment size provided
..0. .... LIBSTACK disposition not provided
..1. .... LIBSTACK disposition provided
...0 0000 reserved

```

```

Byte 1
0... .... LIBSTACK disposition KEEP
1... .... LIBSTACK disposition FREE
.000 0000 reserved

```

In byte 0, there is a flag for each LIBSTACK suboption. The exit must set the flags in byte 0 to indicate the LIBSTACK suboptions for which it is providing values.

***LIBSTACK-init-size* (OUTPUT)**

A fullword binary field used to indicate the number of bytes for the LIBSTACK initial size.

***LIBSTACK-incr-size* (OUTPUT)**

A fullword binary field used to indicate the number of bytes for the LIBSTACK increment size.

***HEAP-flags* (OUTPUT)**

A 2-byte field containing flags. The layout of these flags is as follows:

```

Byte 0
0... .... HEAP initial size not provided
1... .... HEAP initial size provided
.0.. .... HEAP increment size not provided
.1.. .... HEAP increment size provided
..0. .... HEAP location not provided
..1. .... HEAP location provided
...0 .... HEAP disposition not provided
...1 .... HEAP disposition provided
.... 0... HEAP initial size 24 not provided
.... 1... HEAP initial size 24 provided
.... .0.. HEAP increment size 24 not provided
.... .1.. HEAP increment size 24 provided
.... ..00 reserved

Byte 1
0... .... HEAP location ANYWHERE
1... .... HEAP location BELOW
.0.. .... HEAP disposition FREE
.1.. .... HEAP disposition KEEP
..00 0000 reserved

```

In byte 0, there is a flag for each HEAP suboption. The exit must set the flags in byte 0 to indicate the HEAP suboptions for which it is providing values.

***HEAP-init-size* (OUTPUT)**

A fullword binary field used to indicate the number of bytes for the HEAP initial size.

***HEAP-incr-size* (OUTPUT)**

A fullword binary field used to indicate the number of bytes for the HEAP increment size.

***HEAP-init-size24* (OUTPUT)**

A fullword binary field used to indicate the number of bytes for the HEAP initial size for the heap storage obtained below the 16-MB line for applications with ANYWHERE in the HEAP runtime option.

***HEAP-incr-size24* (OUTPUT)**

A fullword binary field used to indicate the number of bytes for the HEAP increment size for the heap storage increments obtained below the 16-MB line for applications with ANYWHERE in the HEAP runtime option.

***ANYHEAP-flags* (OUTPUT)**

A 2-byte field containing flags. The layout of these flags is as follows:

```

Byte 0
0... .... ANYHEAP initial size not provided
1... .... ANYHEAP initial size provided
.0.. .... ANYHEAP increment size not provided
.1.. .... ANYHEAP increment size provided
..0. .... ANYHEAP location not provided
..1. .... ANYHEAP location provided
...0 .... ANYHEAP disposition not provided
...1 .... ANYHEAP disposition provided
.... 0000 reserved

Byte 1
0... .... ANYHEAP location ANYWHERE
1... .... ANYHEAP location BELOW
.0.. .... ANYHEAP disposition KEEP
.1.. .... ANYHEAP disposition FREE
..00 0000 reserved

```

In byte 0, there is a flag for each ANYHEAP suboption. The exit must set the flags in byte 0 to indicate the ANYHEAP suboptions for which it is providing values.

ANYHEAP-init-size (OUTPUT)

A fullword binary field used to indicate the number of bytes for the ANYHEAP initial size.

ANYHEAP-incr-size (OUTPUT)

A fullword binary field used to indicate the number of bytes for the ANYHEAP increment size.

BELOWHEAP-flags (OUTPUT)

A 2-byte field containing flags. The layout of these flags is as follows:

```

Byte 0
0... .... BELOWHEAP initial size not provided
1... .... BELOWHEAP initial size provided
.0.. .... BELOWHEAP increment size not provided
.1.. .... BELOWHEAP increment size provided
..0. .... BELOWHEAP disposition not provided
..1. .... BELOWHEAP disposition provided
...0 0000 reserved

Byte 1
0... .... BELOWHEAP disposition KEEP
1... .... BELOWHEAP disposition FREE
.000 0000 reserved

```

In byte 0, there is a flag for each BELOWHEAP suboption. The exit must set the flags in byte 0 to indicate the BELOWHEAP suboptions for which it is providing values.

BELOWHEAP-init-size (OUTPUT)

A fullword binary field used to indicate the number of bytes for the BELOWHEAP initial size.

BELOWHEAP-incr-size (OUTPUT)

A fullword binary field used to indicate the number of bytes for the BELOWHEAP increment size.

Figure 13 on page 212 shows a mapping of the CEESTX storage used control block. (See Figure 12 on page 208 for a mapping of the CEESTX storage input control block.)

+0	STACK-init-size
+4	STACK-incr-size
+8	STACK-max
+C	STACK-largest-by-thread
+10	STACK-segments-alloc
+14	STACK-segments-freed
+18	LIBSTACK-init-size
+1C	LIBSTACK-incr-size
+20	LIBSTACK-max
+24	LIBSTACK-largest-by-thread
+28	LIBSTACK-segments-alloc
+2C	LIBSTACK-segments-freed
+30	HEAP-init-size
+34	HEAP-incr-size
+38	HEAP-total-used
+3C	HEAP-get-requests
+40	HEAP-free-requests
+44	HEAP-segments-alloc
+48	HEAP-segments-freed
+4C	HEAP24-init-size
+50	HEAP24-incr-size
+54	HEAP24-total-used
+58	HEAP24-get-requests
+5C	HEAP24-free-requests
+60	HEAP24-segments-alloc
+64	HEAP24-segments-freed
+68	ANYHEAP-init-size
+6C	ANYHEAP-incr-size
+70	ANYHEAP-total-used
+74	ANYHEAP-get-requests
+78	ANYHEAP-free-requests
+7C	ANYHEAP-segments-alloc
+80	ANYHEAP-segments-freed
+84	BELOWHEAP-init-size
+88	BELOWHEAP-incr-size
+8C	BELOWHEAP-total-used
+90	BELOWHEAP-get-requests
+94	BELOWHEAP-free-requests
+98	BELOWHEAP-segments-alloc
+9C	BELOWHEAP-segments-freed

Figure 13. CEESTX storage used control block map

The CEESTX storage output control block elements are described as follows:

- Each field is a fullword binary field that corresponds to the information in the Language Environment storage report.

+0	STACK-init-size
+4	STACK-incr-size
+8	STACK-max-allocated
+C	LIBSTACK-init-size
+10	LIBSTACK-incr-size
+14	LIBSTACK-max-allocated
+18	HEAP-init-size
+1C	HEAP-incr-size
+20	HEAP-max-allocated
+24	HEAP24-init-size
+28	HEAP24-incr-size
+2C	HEAP24-max-allocated
+30	ANYHEAP-init-size
+34	ANYHEAP-incr-size
+38	ANYHEAP-max-allocated
+3C	BELOWHEAP-init-size
+40	BELOWHEAP-incr-size
+44	BELOWHEAP-max-allocated

Figure 14. CEESTX storage allocated control block map

The CEESTX storage used control block elements are described as follows:

- Each field is a fullword binary field.
- The init-size fields indicate the initial size specified in the runtime option for the storage area.
- The incr-size fields indicate the increment size specified in the runtime option for the storage area.
- The max-allocated fields indicate the maximum amount of storage allocated for the storage area.

Usage notes

- The storage tuning user exit must be written in assembler and must be reentrant. If you write the storage tuning user exit in Language Environment-enabled assembler, you must specify MAIN=NO on the CEEENTRY macro.
- The storage tuning user exit must not call any HLL programs.
- The storage tuning user exit must not create a Language Environment enclave.
- All values provided by the storage tuning user exit that are not valid for initial size and increment size are ignored. For example, a negative increment size is ignored.
- The storage tuning user exit should provide storage sizes that are multiples of 8. Any storage size that is not a multiple of 8 will be rounded up to the nearest multiple of 8 bytes.
- The STACK location setting cannot be set to ANY when ALL31 is OFF, or it will be ignored.
- Only the following CICS commands can be used during the new load module function: GETMAIN, FREEMAIN, ENQUEUE, DEQUEUE, and any command that performs I/O to a VSAM file, a CICS data table, a transient data queue, or a temporary storage queue.
- Only the following CICS commands can be used during enclave initialization and enclave termination: GETMAIN, FREEMAIN, LOAD, DELETE, ENQUEUE, DEQUEUE, INQUIRE, SET, and any command that performs I/O to a VSAM file, a CICS data table, a transient data queue, or a temporary storage queue.
- CICS considerations: CICS commands can be used in the storage tuning user exit. However, the storage tuning user exit must adhere to the following conventions when using EXEC CICS commands:
 - The storage tuning user exit has to use the CICS system EIB (the SYSEIB translator option must be used).
 - The CICS commands must use the RESP option.
 - Only the following CICS commands can be used during region initialization and region termination: GETMAIN, FREEMAIN, LOAD, DELETE, and any command that performs I/O to a VSAM file, a CICS data table, or a temporary storage queue.
 - The storage tuning user exit cannot use any Language Environment services when called for region initialization, new load module, enclave initialization, and region termination.
- Upon return from the storage tuning user exit, Language Environment takes no action other than continuing with its processing, as no return codes are defined.
- The values from the storage tuning user exit are ignored for those options that are installed as nonoverrideable.
- Register conventions for the storage tuning user exit are:

Register 1

Points to a word which contains the address of the storage tuning user exit control block.

Register 12

Points to the CAA.

When the storage tuning user exit is called for enclave initialization and enclave termination, the CAA is fully initialized.

When the storage tuning user exit is called for region initialization, new load module, and region termination, a partially initialized CAA is provided to enable Language Environment stack processing.

Register 13

Points to a dynamic save area (DSA). The exit routine can save the registers here across its processing.

Register 14

Contains the return address.

Register 15

Contains the entry point address upon entry.

AMODE

The storage tuning user exit is called in AMODE(31) and it must return in AMODE(31).

- The behavior of the RPTSTG option is not affected by the storage tuning user exit. The storage tuning user exit does not cause a Language Environment storage report to be generated.

Appendix B. Using Fortran with Language Environment

This topic provides information for tuning and customizing your Language Environment Fortran runtime routines within Language Environment. The customization information is intended to help you enhance system performance and provide certain I/O characteristics.

The following sections are included:

- “Customizing for Fortran applications link-edited with Language Environment”
- “Customizing for Fortran applications link-edited with VS FORTRAN” on page 223
- “Customizing Fortran LIBPACKS” on page 238

Customizing for Fortran applications link-edited with Language Environment

This section provides information about how to customize Language Environment for Fortran applications that are link-edited with Language Environment. You can customize, or not customize:

- Unit Attribute Table default values (See “Changing the unit attribute table default values.”)
- Language Environment runtime options (See Chapter 5, “Customizing Language Environment runtime options,” on page 19.)

For information about customizing Language Environment if you have Fortran applications that were link-edited with VS FORTRAN Version 1 or 2 for running in load mode, see “Customizing for Fortran applications link-edited with VS FORTRAN” on page 223.

Changing the unit attribute table default values

Module AFHOUTAG contains the Unit Attribute Table defaults and DCB information for each I/O unit. You can accept the IBM-supplied defaults, shown in Figure 15 on page 221, or you can supply your own defaults. To customize AFHOUTAG for your site, use the IBM-supplied job AFHWEUAT, and modify the AFHOUTCM, AFHOUNTM, and AFHODCBM macro instructions in an SMP/E USERMOD. The following sections describe the syntax and operands of the macro instructions.

Starting the unit attribute table definition using the AFHOUTCM macro

Use the AFHOUTCM macro to start and to end the Unit Attribute Table definition. In addition, you can specify default values for information required by the runtime input/output routines of the Fortran component of Language Environment. This section shows the syntax of the operands used for starting the Unit Attribute Table definition.

The syntax of AFHOUTCM macro instruction is as follows:

```
AFHOUTAG AFHOUTCM [UNTABLE={ highunit | 99 } ]  
[,DEVICE={ device-name | SYSDA } ]
```

UNTABLE=*highunit*

Specifies the largest unit number that can be used in any Fortran program in I/O statements other than the CLOSE and INQUIRE statements. *highunit* must be an integer between 8 and 2000, inclusive. If the UNTABLE parameter is omitted, the default value of *highunit* is 99.

DEVICE=*device-name*

Specifies where dynamically allocated data sets are placed if there is no overriding value given through an invocation of the FILEINF callable service. *device-name* can be a unit address, a group name, or a device type for a DASD device. A unit address is 3 or 4 hexadecimal digits consisting of the channel, control unit, and device number. A group name is any name that is defined during MVS system generation for a DASD device such as SYSDA or DISK. The device type is the IBM-supplied name such as 3380 or 3390.

If the DEVICE parameter is omitted, the default value is SYSDA.

Associating units with DCB characteristics using the AFHOUNTM macro

Use the AFHOUNTM macro to specify a single unit, or group of units, that is to be associated with a set of DCB default values. Use the AFHOUNTM macro in conjunction with the AFHODCBM macro.

Syntax of AFHOUNTM macro instruction

AFHOUNTM { *unitno* | (*unitno*, *qty*) | RDRUNIT | PRTUNIT | PUNUNIT }
,DCBSET=*label*

unitno

The unit number, or the first in a series of consecutive unit numbers, for which the set of default DCB characteristics referenced by the DCBSET parameter is to be applied. If *unitno* is the number of the error message unit (or if the error message unit is included in the range covered by *qty*, following), the specification is ignored for the error message unit.

qty

The number of consecutive unit numbers, beginning with *unitno*, for which the set of default DCB characteristics referenced by the DCBSET parameter is to be applied.

RDRUNIT

Indicates that the set of default DCB characteristics referenced by the DCBSET parameter is to be applied to the standard input unit. The *standard input unit* is the unit to which a READ statement applies when the unit identifier is given as *. The number of the standard input unit is the value given by the RDRUNIT runtime option or its default.

Even though there may also be an AFHOUNTM macro instruction that refers to the standard input unit by its unit number (that is, with the *unitno* form of specification), the AFHOUNTM with the RDRUNIT parameter takes precedence and applies to the standard input unit.

If there is no AFHOUNTM macro instruction with a RDRUNIT parameter, then the default DCB characteristics for the standard input unit are those referenced by an AFHOUNTM macro instruction that refers to this unit with the *unitno* form of specification.

PRTUNIT

Indicates that the set of default DCB characteristics referenced by the DCBSET parameter is to be applied to the print unit.

The *print unit* is one of the standard output units and is the unit to which either a WRITE statement with a unit identifier of * or a PRINT statement applies. The number of the print unit is the value given by the PRTUNIT runtime option or its default if the number of the print unit is different than the number of the error message unit.

The *error message unit* the unit to which output such as error messages and dumps from services such as CDUMP and SDUMP is directed. The number of the error message unit is the value given by the ERRUNIT runtime option or its default.

The *punch unit* is one of the standard output units and is the unit to which a PUNCH statement applies. The number of the punch unit is the value given by the PUNUNIT runtime option or its default.

Even though there may also be an AFHOUNTM macro instruction that refers to the print unit by its unit number (that is, with the *unitno* form of specification), the AFHOUNTM with the PRTUNIT parameter takes precedence and applies to the print unit.

If there is no AFHOUNTM macro instruction with a PRTUNIT parameter and if the print unit and the error message units are different units, then the default DCB characteristics for the print unit are those referenced by an AFHOUNTM macro instruction that refers to this unit with the *unitno* form of specification.

PUNUNIT

Indicates that the set of default DCB characteristics referenced by the DCBSET parameter is to be applied to the punch unit. The *punch unit* is one of the standard output units and is the unit to which a PUNCH statement applies. The number of the punch unit is the value given by the PUNUNIT runtime option or its default.

Even though there may also be an AFHOUNTM macro instruction that refers to the punch unit by its unit number (that is, with the *unitno* form of specification), the AFHOUNTM with the PUNUNIT parameter takes precedence and applies to the punch unit.

If there is no AFHOUNTM macro instruction with a PUNUNIT parameter, then the default DCB characteristics for the punch unit are those referenced by a AFHOUNTM macro instruction that refers to this unit with the *unitno* form of specification.

DCBSET=*label*

The identifier of the DCB attributes to associate with this unit, set of units, or standard I/O unit. This is the name given in the associated AFHODCBM macro instruction.

Specifying the DCB characteristics using the AFHODCBM macro

Use the AFHODCBM macro to specify default DCB information for the I/O units that have a DCBSET=*label* parameter on the AFHOUNTM macro.

The syntax of AFHODCBM macro instruction is as follows:

```
[label] AFHODCBM [,SFBUFNO=number | 2]  
[SUBUFNO=number | 2]  
[SFBLKSI=number | 800]  
[SUBLKSI=number | 800]  
[SFLRECL=number | 800]  
[SULRECL=number | -1]
```

[,SFRECFM=*char* | U]
[,SURECFM=*char* | VS]
[,SFMAXRE=*number* | 100]
[,SUMAXRE=*number* | 100]
[,DMAXRE=*number* | 100]

label

The name specified in the DCBSET parameter of one or more AFHOUNTM macro instructions to relate the I/O units to this set of DCB default values.

If *label* is omitted, the DCB data is assigned to all units defined in the Unit Attribute Table by the AFHOUTCM macro instruction that do not have a AFHOUNTM macro instruction. If any of the units in the Unit Attribute Table do not have their own AFHOUNTM macro instruction, then you must provide a AFHODCBM macro instruction **without a label** to apply defaults to these units.

SFBUFNO=*number* | 2

Specifies the default value for the number of buffers for sequential formatted files on DASD or tape. *number* must be a value greater than or equal to 1 and less than or equal to 255.

SUBUFNO=*number* | 2

Specifies the default value for the number of buffers for sequential unformatted files on DASD or tape. *number* must be a value greater than or equal to 1 and less than or equal to 255.

SFBLKSI = *number* | 800

Specifies the block size for sequential formatted files. *number* is an integer expression of length 4 bytes; valid range of the blocksize is from 1 to 32760.

SUBLKSI = *number* | 800

Specifies the block size for sequential unformatted files. *number* is an integer expression of length 4 bytes; valid range of the blocksize is from 1 to 32760.

SFLRECL = *number* | 800

Specifies the logical record length for sequential formatted files. *number* is an integer expression of length 4 bytes; valid range is from 1 to 32756 for variable record formats (SURECFM= V, VA, VB, or VBA), or 1 to 32760 for all other record formats.

SULRECL = *number* | -1

Specifies the logical record length for sequential unformatted files. *number* is an integer expression of length 4 bytes; valid range is from 1 to 32756 for variable record formats (SURECFM= V, VA, VB, VBA, VS, or VBS), or 1 to 32760 for all other record formats or -1, which specifies an unlimited record length. -1 is valid for SURECFM=VS or VBS formats.

SFRECFM = *char* | U

Specifies the record format for sequential formatted files. The value of *char* must be F, FA, FB, FBA, V, VA, VB, VBA, U, or UA. For more information about I/O, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS*

SURECFM = *char* | VS

Specifies the record format for sequential unformatted files. The value of *char* must be F, FA, FB, FBA, V, VA, VB, VBA, VS, VBS, U, or UA. For more information about I/O, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS*

SFMAXRE = *number* | 100

Specifies the amount of space to be converted into blocks in a sequential

formatted file. It is only valid for new DASD files; if specified for an existing file, it will be ignored. *number* is an integer expression of length 4. See MAXREC in *VS FORTRAN Version 2 Programming Guide for CMS and MVS* for information about how space is converted to blocks.

SUMAXRE = *number* | 100

Specifies the amount of space to be converted into blocks in a sequential unformatted file. It is only valid for new DASD files; if specified for an existing file, it will be ignored. *number* is an integer expression of length 4. See MAXREC in *VS FORTRAN Version 2 Programming Guide for CMS and MVS* for information about how space is converted to blocks.

DMAXRE = *number* | 100

Specifies the amount of space to be converted into blocks in a direct file. It is only valid for new DASD files; if specified for an existing file, it will be ignored. *number* is an integer expression of length 4. See *VS FORTRAN Version 2 Programming Guide for CMS and MVS* for information about how space is converted to blocks.

CAUTION: If you change the IBM-supplied default DCB values, any existing Fortran programs that depend on the original defaults might not work.

Ending the unit attribute table definition using the AFHOUTCM macro

The AFHOUTCM macro is used to start and to end the Unit Attribute Table definition. This section shows the syntax of the operands used for ending the Unit Attribute Table definition.

The syntax of **AFHOUTCM macro instruction: Final statement** is as follows:

AFHOUTCM TYPE=FINAL

IBM-supplied unit attribute table default values

The macro instructions shown in Figure 15 are provided in the module AFHOUTAG. This module is used to set up the IBM-supplied default values for the standard I/O units, and file characteristics such as the DCB information.

```
AFHOUTAG AFHOUTCM UNTABLE=99,  
        DEVICE=SYSDA  
  
        AFHOUNTM RDRUNIT,DCBSET=DCBRDR  
        AFHOUNTM PRTUNIT,DCBSET=DCBPRT  
        AFHOUNTM PUNUNIT,DCBSET=DCBPUN  
  
DCBRDR AFHODCBM SFRECFM=F,SFLRECL=80,SFBLKSI=80,  
        SURECFM=F,SULRECL=80,SUBLKSI=80  
  
DCBPRT AFHODCBM SFRECFM=UA,SFLRECL=133,SFBLKSI=133  
  
DCBPUN AFHODCBM SFRECFM=F,SFLRECL=80,SFBLKSI=80,  
        SURECFM=F,SULRECL=80,SUBLKSI=80  
  
        AFHODCBM SFRECFM=U,SFLRECL=800,SFBLKSI=800,SFMAXRE=100,  
        SURECFM=VS,SULRECL=-1,SUBLKSI=800,SUMAXRE=100,  
        DMAXRE=100  
  
AFHOUTCM TYPE=FINAL
```

Figure 15. IBM-supplied macro instructions

Note: The format of this particular example is given for readability purposes. Remember to add the necessary continuation flags in column 72, and to begin continued lines in column 16.

The three AFHOUNTM macro instructions indicate that the standard input unit, the print unit, and the punch unit have the default DCB information provided on the first three AFHODCBM macro instructions. Note that the last AFHODCBM macro does not have a label; its set of defaults apply to all units except the standard I/O units. For more information about the RDRUNIT, ERRUNIT, PRTUNIT, and PUNUNIT runtime options, which are used to specify the unit numbers of these standard I/O units, see *z/OS Language Environment Programming Reference*.

Examples of changing unit attribute table default values

The following example shows how you can modify the IBM-supplied defaults for your own environment. You can alter instructions by typing over existing data, or you can remove or add AFHOUNTM and AFHODCBM macro instructions.

Example

In this example, the device name SYSSQ is specified and a unique set of DCB attributes is assigned to units 1 through 4 for dynamically allocated data sets.

```
AFHOUTAG AFHOUTCM UNTABLE=99,  
          DEVICE=SYSSQ  
  
          AFHOUNTM RDRUNIT,DCBSET=DCBRDR  
          AFHOUNTM PRTUNIT,DCBSET=DCBPRT  
          AFHOUNTM PUNUNIT,DCBSET=DCBPUN  
          AFHOUNTM (1,4),DCBSET=USERDCB  
  
DCBRDR AFHODCBM SFRECFM=F,SFLRECL=80,SFBLKSI=80,  
          SURECFM=F,SULRECL=80,SUBLKSI=80  
  
DCBPRT AFHODCBM SFRECFM=UA,SFLRECL=133,SFBLKSI=133  
  
USERDCB AFHODCBM SFRECFM=FB,SFLRECL=50,SFBLKSI=250,  
          SFMAXRE=200,SURECFM=FB,SULRECL=50,  
          SUBLKSI=250,SUMAXRE=200,DMAXRE=200  
  
DCBPUN AFHODCBM SFRECFM=F,SFLRECL=80,SFBLKSI=80,  
          SURECFM=F,SULRECL=80,SUBLKSI=80  
  
          AFHODCBM SFRECFM=U,SFLRECL=800,SFBLKSI=800,SFMAXRE=100,  
          SURECFM=VS,SULRECL=-1,SUBLKSI=800,SUMAXRE=100,  
          DMAXRE=100  
  
AFHOUTCM TYPE=FINAL
```

Figure 16. Modified IBM-supplied macro instructions

Note: The format of the example is given for readability purposes. Remember to add the necessary continuation flags in column 72, and to begin continued lines in column 16.

AFHOUTCM, AFHOUNTM, and AFHODCBM must all be coded, in that order, followed by the AFHOUTCM TYPE=FINAL statement.

Customizing for Fortran applications link-edited with VS FORTRAN

This section contains information about how to customize Language Environment if you have Fortran applications that were link-edited with VS FORTRAN Version 1 or 2 for running in load mode.

If you have such applications, you can customize, or not customize:

- VS FORTRAN Unit Attribute Table defaults (See “Changing the unit attribute table default values” following.)
- VS FORTRAN runtime option defaults (See “Changing VS FORTRAN runtime option defaults” on page 229.)
- VS FORTRAN Error Option Table defaults (See “Changing the error option table defaults” on page 234.)

Note: Language Environment provides a VS FORTRAN compatibility library for running Fortran applications that are not link-edited with Language Environment.

You can customize Language Environment to provide certain runtime characteristics for Fortran applications that were link-edited with VS FORTRAN for running in load mode. You use macros with the same names as you used in VS FORTRAN Version 2 Release 6. These macros are VSF2UAT, VSF2UNIT, VSF2DCB, VSF2PARAM, and VSF2UOPT. Each of these macros is available in Language Environment with these macro names as aliases for members with names beginning with AFH5. The use of these macros is identical to that in VS FORTRAN Version 2 Release 6; therefore, if you have assembler language source files that you used in the past, you can use these same source files to customize Language Environment.

Changing the unit attribute table default values

Module AFH5VUAT contains the Unit Attribute Table defaults and DCB information for each I/O unit of the VS FORTRAN compatibility library. You can accept the IBM-supplied defaults, shown in Figure 17 on page 227, or you can supply your own defaults. To customize AFH5VUAT for your site, use the IBM-supplied job AFHWVUAT, and modify the VSF2UAT, VSF2UNIT, and VSF2DCB macro instructions in an SMP/E USERMOD. The following sections describe the syntax and operands of the macro instructions.

Starting the unit attribute table definition using the VSF2UAT macro

The VSF2UAT macro is used to start and to end the Unit Attribute Table definition. Fortran component of Language Environment. In addition, you can specify default values for information required by the runtime input/output routines of the VS FORTRAN compatibility library. This section shows the syntax of the operands used for starting the Unit Attribute Table definition.

Syntax of VSF2UAT Macro: Statement Form

```
[name] VSF2UAT [DECIMAL=PERIOD | COMMA]
[,PUNCH=number | 7 ]
[,ERRMSG=number | 6 ]
[,PRINTER=number | 6 ]
[,READER=number | 5 ]
[,UNTABLE=number | 99]
[,DEVICE=device-name | SYSDA]
```

See “Ending the unit attribute table definition using the VSF2UAT macro” on page 227 for the form of VSF2UAT as the final macro instruction.

The IBM-supplied default values are underlined in the following option list. If an option is not specified, its default value will be used.

name

Specifies a name, such as AFBVUAT or AFH5UAT. *name* is ignored, and the CSECT name becomes AFH5VUAT automatically.

DECIMAL = PERIOD | COMMA

Specifies the character to be used as the decimal indicator in printed output.

PUNCH = *number* | 7

Specifies, for L^{ANGLVL}(66) only, the standard I/O unit number for the PUNCH statement to send data to the card punch. The number specified must be between 0 and 99 or the value specified for the UNTABLE parameter, for UNTABLE values less than or equal to 99. It must not be the same as the number specified for ERRMSG, PRINTER, or READER.

ERRMSG = *number* | 6

Specifies the standard I/O unit number for the error messages generated by VS FORTRAN Version 2 Library. The number specified must be between 0 and 99 or the value specified for the UNTABLE parameter, for UNTABLE values less than or equal to 99. It must not be the same as the number specified for PUNCH or READER; it can be the same number specified for PRINTER.

PRINTER = *number* | 6

Specifies the standard I/O unit number for the print statement, and with any WRITE statement specifying an installation-dependent form of the unit. The number specified must be between 0 and 99 or the value specified for the UNTABLE parameter, for UNTABLE values less than or equal to 99. It must not be the same as that specified for PUNCH and READER. It can be the same number specified for ERRMSG.

READER = *number* | 5

Specifies the standard I/O unit number for any READ statement specifying an installation-dependent form of the unit. The number specified must be between 0 and 99 or the value specified for the UNTABLE parameter, for UNTABLE values less than or equal to 99. It must not be the same as the number specified for either PUNCH, ERRMSG, or PRINTER.

UNTABLE = *number* | 99

Specifies the largest unit number you can include in a VS FORTRAN program. It can be specified as any integer between 8 and 2000.

DEVICE = *device-name* | SYSDA

Specifies where dynamically allocated data sets are placed if there is no overriding value given through an invocation of the FILEINF callable service. *device-name* can be a unit address, a group name, or a device type for a DASD device. A unit address is 3 or 4 hexadecimal digits consisting of the channel, control unit, and device number. A group name is any name that is defined during MVS system generation for a DASD device such as SYSDA or DISK. The device type is the IBM-supplied name such as 3380 or 3390.

If the DEVICE parameter is omitted, the default value is SYSDA.

Note: In Fortran, the units described by the PUNCH, ERRMSG, PRINTER and READER parameters are called standard I/O units.

Associating units with dcb characteristics using the VSF2UNIT macro

Use the VSF2UNIT macro to specify a single unit, or group of units, that is to be associated with a set of DCB default values. The VSF2UNIT macro is used in conjunction with the VSF2DCB macro.

Syntax of VSF2UNIT Macro

```
VSF2UNIT { unitno | ( unitno [qty] ) } ,DCBSET = label
```

unitno

Specifies the unit number, or the first in a series of consecutive unit numbers, that are to have DCB default values assigned.

qty

Specifies, if there is more than one, the number of consecutive unit numbers, beginning with *unitno*, that are to have DCB default values assigned.

DCBSET=*label*

Specifies the identifier of the DCB attributes to associate with this unit or set of units. This is the name given in the associated VSF2DCB macro instruction.

Specifying the DCB characteristics using the VSF2DCB macro

Use the VSF2DCB macro to specify DCB default information for the I/O units that have DCBSET=*label* parameter of the VSF2UNIT macro.

The syntax of VSF2DCB macro is as follows:

```
[label] VSF2DCB [,SFBUFNO=number | 2]  
[,SUBUFNO=number | 2]  
[,SFBLKSI=number | 800]  
[,SUBLKSI=number | 800]  
[,SFLRECL=number | 800]  
[,SULRECL=number. | -1]  
[,SFRECFM=char | U]  
[,SURECFM=char | VS]  
[,SFMAXRE=number | 100]  
[,SUMAXRE=number | 100]  
[,DMAXRE=number | 100]
```

label

Specified in the VSF2UNIT macro to identify the I/O units that are to be assigned DCB default values.

If *label* is omitted, the DCB data is assigned to all units defined in the default table by the VSF2UAT macro, but which have not been defined by the VSF2UNIT macro. If any of the units defined in the attribute table do not have their own associated DCBSET coded, you must provide a VSF2DCB macro **without a label** to apply defaults to these units.

SFBUFNO=*number* | 2

Specifies the default value for the number of buffers for sequential formatted files on DASD or tape. *number* must be a value greater than or equal to 1 and less than or equal to 255.

SUBUFNO=*number* | 2

Specifies the default value for the number of buffers for sequential unformatted files on DASD or tape. *number* must be a value greater than or equal to 1 and less than or equal to 255.

SFBLKSI = number | 800

Specifies the block size for sequential formatted files. *number* is an integer expression of length 4 bytes; valid range of the blocksize is from 1 to 32760.

SUBLKSI = number | 800

Specifies the block size for sequential unformatted files. *number* is an integer expression of length 4 bytes; valid range of the blocksize is from 1 to 32760.

SFLRECL = number | 800

Specifies the logical record length for sequential formatted files. *number* is an integer expression of length 4 bytes; valid range is from 1 to 32756 for variable record formats (SURECFM= V, VA, VB, or VBA), or 1 to 32760 for all other record formats.

SULRECL = number | -1

Specifies the logical record length for sequential unformatted files. *number* is an integer expression of length 4 bytes; valid range is from 1 to 32756 for variable record formats (SURECFM= V, VA, VB, VBA, VS, or VBS), or 1 to 32760 for all other record formats or -1, which specifies an unlimited record length. -1 is valid for SURECFM=VS or VBS formats.

SFRECFM = char | U

Specifies the record format for sequential formatted files. The value of *char* must be F, FA, FB, FBA, V, VA, VB, VBA, U, or UA. For more information about I/O, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS*

SURECFM = char | VS

Specifies the record format for sequential unformatted files. The value of *char* must be F, FA, FB, FBA, V, VA, VB, VBA, VS, VBS, U, or UA. For more information about I/O, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS*

SFMAXRE = number | 100

Specifies the amount of space to be converted into blocks in a sequential formatted file. It is only valid for new DASD files; if specified for an existing file, it will be ignored. *number* is an integer expression of length 4. See MAXREC in *VS FORTRAN Version 2 Programming Guide for CMS and MVS* for information about how space is converted to blocks.

SUMAXRE = number | 100

Specifies the amount of space to be converted into blocks in a sequential unformatted file. It is only valid for new DASD files; if specified for an existing file, it will be ignored. *number* is an integer expression of length 4. See MAXREC in *VS FORTRAN Version 2 Programming Guide for CMS and MVS* for information about how space is converted to blocks.

DMAXRE = number | 100

Specifies the amount of space to be converted into blocks in a direct file. It is only valid for new DASD files; if specified for an existing file, it will be ignored. *number* is an integer expression of length 4. See *VS FORTRAN Version 2 Programming Guide for CMS and MVS* for information about how space is converted to blocks.

CAUTION: If you change the IBM-supplied default DCB values, the existing Fortran programs that depend on the original defaults may not work. For more information about DCB values, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS* .

Ending the unit attribute table definition using the VSF2UAT macro

Use the VSF2UAT macro to start and to end the Unit Attribute Table definition. Use the following form of VSF2UAT as the final macro instruction in the Unit Attribute Table definition.

Syntax of VSF2UAT Macro: Final Statement

```
VSF2UAT TYPE=FINAL
```

TYPE = FINAL

Is the required last statement of the VSF2UAT macro.

IBM-supplied unit attribute table default values

The macro instructions shown in Figure 17 are provided in the module AFH5VUAT. This module is used to set up the IBM-supplied default values for the standard I/O units, and file characteristics such as the DCB information.

```
AFH5VUAT VSF2UAT UNTABLE=99,
                DECIMAL=PERIOD,
                READER=5,
                ERRMSG=6,
                PRINTER=6,
                PUNCH=7,
                DEVICE=SYSDA

                VSF2UNIT 5,DCBSET=DCBRDR
                VSF2UNIT 6,DCBSET=DCBPRT
                VSF2UNIT 7,DCBSET=DCBPUN

DCBRDR VSF2DCB SFRECFM=F,SFLRECL=80,SFBLKSI=80,
                SURECFM=F,SULRECL=80,SUBLKSI=80

DCBPRT VSF2DCB SFRECFM=UA,SFLRECL=133,SFBLKSI=133

DCBPUN VSF2DCB SFRECFM=F,SFLRECL=80,SFBLKSI=80,
                SURECFM=F,SULRECL=80,SUBLKSI=80

                VSF2DCB SFRECFM=U,SFLRECL=800,SFBLKSI=800,SFMAXRE=100,
                SURECFM=VS,SULRECL=-1,SUBLKSI=800,SUMAXRE=100,
                DMAXRE=100

                VSF2UAT TYPE=FINAL
```

Figure 17. IBM-supplied macro instructions

Note: The above format is given for readability purposes. Remember to add the necessary continuation flags in column 72, and to begin continued lines in column 16.

The three VSF2UNIT macro instructions indicate that units 5, 6, and 7 have the default DCB information provided on the first three VSF2DCB macro instructions. Note that the last VSF2DCB macro does not have a label; its set of defaults apply to all units except 5, 6, and 7. For more information about the RDRUNIT, ERRUNIT, PRTUNIT, and PUNUNIT runtime options, which are used to specify the unit numbers of these standard I/O units, see *VS FORTRAN Version 2 Programming Guide for CMS and MVS*.

Examples of changing unit attribute table default values

The following examples show how you can modify the IBM-supplied defaults for your own environment. You can alter instructions by typing over existing data, or you can add more VSF2UNIT and VSF2DCB macro instructions.

Example 1

In this example, we have specified device name SYSSQ for dynamically allocated data sets and assigned a unique set of DCB attributes to units 1 through 4. The DCB Information for both sequential formatted and unformatted files written on these units is indicated in the first VSF2DCB macro instruction (“USERDCB”) shown in Figure 18.

AFH5VUAT	VSF2UAT	DEVICE= SYSSQ
	VSF2UNIT	(1,4),DCBSET= USERDCB
	VSF2UNIT	5,DCBSET=DCBRDR
	VSF2UNIT	6,DCBSET=DCBPRT
	VSF2UNIT	7,DCBSET=DCBPUN
USERDCB	VSF2DCB	SFRECFM=FB,SFLRECL=50,SFBLKSI=250,SFMAXRE=200, SURECFM=FB,SULRECL=50,SUBLKSI=250,SUMAXRE=200, DMAXRE=200
DCBRDR	VSF2DCB	SFRECFM=F,SFLRECL=80,SFBLKSI=80, SURECFM=F,SULRECL=80,SUBLKSI=80
DCBPRT	VSF2DCB	SFRECFM=UA,SFLRECL=133,SFBLKSI=133
DCBPUN	VSF2DCB	SFRECFM=F,SFLRECL=80,SFBLKSI=80, SURECFM=F,SULRECL=80,SUBLKSI=80
	VSF2DCB	SFRECFM=U,SFLRECL=800,SFBLKSI=800,SFMAXRE=100, SURECFM=VS,SULRECL=-1,SUBLKSI=800,SUMAXRE=100, DMAXRE=100
	VSF2UAT	TYPE=FINAL

Figure 18. Modified IBM-supplied macro instructions

Note: The above format is given for readability purposes. Remember to add the necessary continuation flags in column 72, and to begin continued lines in column 16.

VSF2UAT, VSF2UNIT, and VSF2DCB must all be coded, in that order, followed by the VSF2UAT TYPE=FINAL statement.

Example 2

If you want to change the unit numbers of the standard input unit, the error message unit, the print unit, and the punch unit, to 1, 2, 3, 4, respectively, modify the IBM-supplied macros as shown in Figure 19 on page 229.

AFH5VUAT	VSF2UAT	DECIMAL=PERIOD, READER=1, ERRMSG=2, PRINTER=3, PUNCH=4, DEVICE=SYSDA
	VSF2UNIT	1,DCBSET=DCBRDR
	VSF2UNIT	2,DCBSET=DCBTERM
	VSF2UNIT	3,DCBSET=DCBPRT
	VSF2UNIT	4,DCB=DCBPUN

Figure 19. Modified IBM-supplied macro instructions

Changing VS FORTRAN runtime option defaults

Module AFH5GPRM contains the set of runtime option defaults for running with the VS FORTRAN compatibility library. You can accept the IBM-supplied defaults, shown in this section, or you can supply your own defaults. To customize AFBVGPRM for your site, use the IBM-supplied job AFHWVPRM, and modify the VSF2PARAM macro instruction in an SMP/E USERMOD. The syntax and operands of the VSF2PARAM macro instruction are described in this section.

Use the AFH5PARAM macro to change the IBM-supplied default values for VS FORTRAN runtime options. The default values you assign are assumed if you do not override them.

There are no operands to set the default values for the runtime options AUTOTASK, PARALLEL, and PARTRACE; therefore these options cannot be changed during installation. They can, however, be changed at runtime.

There are no operands in the VSF2PARAM macro to set the default values for the runtime options ERRUNIT, RDRUNIT, PRTUNIT, and PUNUNIT. The default I/O unit values for these units can be changed during installation through the Unit Attribute Table.

Syntax of VSF2PARAM Macro Instruction

```
VSF2PARAM SCOPE = GLOBAL
[ABSDUMP | NOABSDUMP]
[CNVIOERR | NOCNVIOERR]
[DEBUG | NODEBUG]
[DEBUNIT(s1[,s2,...]) | NODEBUNIT]
[ECPACK | NOECPACK]
[FAIL(ABEND | RC | ABENDRC)]
[FILEHIST | NOFILEHIST]
[INQPCOPN | NOINQPCOPN]
[IOINIT | NOIOINIT]
[OCSTATUS | NOOCSTATUS]
[RECPAD[(ALL)] | NORECPAD]
[SPIE | NOSPIE]
[STAE | NOSTAE]
[XUFLOW | NOXUFLOW]
```

The IBM-supplied default values are underlined in the following option list. If an option is not specified, its default will be used, with the exception of the SCOPE option, which must always be specified.

SCOPE = GLOBAL

Required to replace the global runtime options table AFBVGPRM, which supplies default values for all users of the VS FORTRAN compatibility library.

There is no default value for this option. Thus SCOPE=GLOBAL must always be specified.

ABSDUMP | NOABSDUMP

Specifies whether the post-abend symbolic dump information is printed.

ABSDUMP

Causes the post-abend symbolic dump information to be printed in the event of an abnormal termination.

NOABSDUMP

Suppresses the printing of the post-abend symbolic dump information.

CNVIOERR | NOCNVIOERR

Specifies whether input conversion errors will be treated as I/O errors.

CNVIOERR

Causes ERR and IOSTAT to recognize conversion errors as I/O errors.

NOCNVIOERR

Causes conversion errors not to be treated as I/O errors. ERR and IOSTAT have no effect for these errors.

DEBUG | NODEBUG

Specifies whether interactive debug will be invoked.

Note: This option does not apply to the Language Environment VS FORTRAN compatibility library. If you want to use the VS FORTRAN Interactive Debugger, then run your program with the VS FORTRAN Version 2 library rather than with Language Environment.

DEBUNIT | NODEBUNIT

Specifies whether Fortran unit numbers will be treated as if connected to a terminal device.

Note: This option does not apply to the Language Environment VS FORTRAN compatibility library. If you want to use the VS FORTRAN Interactive Debugger, then run your program with the VS FORTRAN Version 2 library rather than with Language Environment.

ECPACK | NOECPACK

Specifies whether a data space should be filled with as many extended common blocks as possible before a new data space is allocated.

ECPACK

Specifies extended common blocks be placed into the fewest possible number of data spaces. This option reduces some of the overhead associated with referencing data spaces.

NOECPACK

Specifies that each extended common block be placed into a separate data space. As a result, reference errors made beyond the bounds of an extended common block might be more easily detected.

FAIL (ABEND | RC | ABENDRC)

Indicates how applications that fail are to be terminated: either by a nonzero return or by an abnormal termination (ABEND). The suboption of the FAIL option may have the following meanings.

ABEND

Causes the program to end by an abnormal termination (ABEND) with a user completion code of 240.

RC Causes the program to end normally but with a nonzero return code (16).

ABENDRC

Causes the program to end by abnormal termination (ABEND) when failure is because of a condition for which the operating system would usually cause an ABEND; and to end with a nonzero return code when failure is by some condition detected by VS FORTRAN.

FILEHIST | NOFILEHIST

Specifies whether to allow the file definition of a file referred to by a ddname to be changed at runtime.

FILEHIST

Causes the history of a file to be used in determining its existence. In particular it checks to see whether:

- The file was ever internally opened (in which case it exists)
- The file was deleted by a CLOSE statement (in which case it does not exist).

When FILEHIST is specified, you cannot change the file definition of a file at runtime and have the same results produced as previous VS FORTRAN releases.

NOFILEHIST

Causes the history of a file to be disregarded in determining its existence.

If you specify NOFILEHIST you should consider:

- **If you change file definitions at runtime:** the file is treated as if it was being opened for the first time. Note that before the file definition can be changed, the existing file must be closed.
- **If you do not change file definitions at runtime:** you must use STATUS='NEW' to re-open an empty file that has been closed with STATUS='KEEP', because the file does not appear to exist to Fortran.

INQPCPN | NOINQPCPN

Specifies whether a unit is connected to a file when executing an INQUIRE by unit.

INQPCPN

Specifies that, if a unit is connected to a file, even if it was preconnected and no I/O statement has been executed, a value of true is returned in the variable or an array element given in the OPENED specifier from an INQUIRE by unit statement.

NOINQPCPN

Indicates that, if and only if a unit is internally open, a value of true is returned in the variable or an array element given in the OPENED specifier for an INQUIRE by unit statement.

"Internally open" means that the unit is connected to a file by an OPEN statement, or if the unit has been preconnected, that a READ, WRITE, PRINT, REWIND, or ENDFILE statement has been successfully executed.

IOINIT | NOIOINIT

Specifies whether the normal initialization for I/O processing will occur during initialization of the runtime environment.

IOINIT

Causes the normal initialization for I/O processing to occur during initialization of the runtime environment.

NOIOINIT

Suppresses initialization for I/O processing. This means that the error message unit is not opened during initialization of the runtime environment. However, this does not prevent I/O from occurring on this or on any other unit. (Such I/O might fail if proper DD statements are not given.)

OCSTATUS | NOOCSTATUS

Specifies whether file existence is checked during the running of OPEN statements, whether files are deleted from their storage media, and whether files that were closed can be reconnected without an OPEN statement.

OCSTATUS

Specifies:

1. File existence is checked for consistency with the OPEN statement specifiers STATUS='OLD' and STATUS='NEW'.
2. File deletion occurs when the CLOSE statement specifier STATUS='DELETE' is given (on devices which allow deletion).
3. A preconnected file is disconnected when a CLOSE statement is given or when another file is opened on the same unit. It can be reconnected only by an OPEN statement when there is no other file currently connected to that unit.

NOOCSTATUS

Specifies:

1. File existence is not checked for consistency with the OPEN statement specifiers STATUS='OLD' and STATUS='NEW'.
2. File deletion does not occur when the CLOSE statement specifier STATUS='DELETE' is given.
3. A preconnected file is disconnected when a CLOSE statement is given or when another file is opened on the same unit. It can be reconnected by a sequential READ or WRITE, BACKSPACE, OPEN, REWIND, or ENDFILE statement when there is no other file currently connected to that unit.

RECPAD[(ALL)] | NORECPAD

Specifies whether a formatted input record is padded with blanks.

RECPAD

Causes a formatted input record within an internal file or a varying/undefined length record (RECFM=U or V) external file to be padded with blanks when an input list and format specification require more data from the record than the record contains. Blanks added for padding are interpreted as though the input record actually contains blanks in those fields. If ALL is specified, a formatted input record is padded regardless of the record format of the file.

NORECPAD

Specifies that an input list and format specification must not require more data from an input record than the record contains. If more data is required, condition FOR1002E is raised.

SPIE | NOSPIE

Specifies whether the runtime environment takes control when a program interrupt occurs.

SPIE

Specifies that the runtime environment takes control when a program interrupt occurs.

NOSPIE

Specifies that the runtime environment does not take control when a program interrupt occurs. If you specify NOSPIE, various runtime functions that depend on a return of control after a program interrupt are not available. These include the following:

- The messages and corrective action for a floating-point overflow
- The messages and corrective action for a floating-point underflow interrupt (unless the underflow is to be handled by the hardware based upon the XUFLOW option)
- The messages and corrective action for a floating-point or fixed-point divide exception
- The simulation of extended precision floating-point operations on processors that do not have these instructions
- The realignment of vector operands that are not on the required storage boundaries and the rerunning of the failed instruction.

Instead of the corrective action, abnormal termination results. In this case, the STAE or NOSTAE option that is in effect governs whether the VS FORTRAN runtime environment gains control at the time of the abend.

STAE | NOSTAE

Specifies whether the runtime environment takes control if an abnormal termination occurs.

STAE

Specifies that the runtime environment will take control when an abnormal termination occurs.

NOSTAE

Specifies that the runtime environment does not take control when an abnormal termination occurs. If NOSTAE is specified, abnormal termination is handled by the operating system rather than by the VS FORTRAN runtime environment. In this case the following occurs:

- Message AFB240I, which shows the PSW and register contents at the time of the abend, is not printed. However, this information will be provided by the operating system.
- The indication of which Fortran statement caused the failure will not be printed.
- The traceback of the routines will not be printed.
- The post-abend symbolic dump will not be printed even with the option ABSDUMP in effect.
- Certain exceptional conditions handled by the runtime environment or by the debugging device cause system abends rather than VS FORTRAN messages. For example, some errors that occur during running of an OPEN statement result in a system abend rather than the printing of message AFB219I, which allows the program to possibly continue running.

- An MTF subtask that terminates unexpectedly causes a user ABEND 922 in the main task rather than message AFB922I.

XUFLOW | NOXUFLOW

Specifies whether an exponent underflow will cause a program interrupt.

XUFLOW

Allows an exponent underflow to cause a program interrupt, followed by a message from the VS FORTRAN Version 2 Library, followed by a standard fixup.

NOXUFLOW

Suppresses the program interrupt caused by an exponent underflow. The hardware sets the result to zero.

Changing the error option table defaults

Module AFH5UOPT contains the Error Option Table defaults. You can accept the IBM-supplied defaults, or you can supply your own defaults. To customize AFH5UOPT for your site, use the IBM-supplied job AFHWVOPT, and modify the VSF2UOPT macro instructions in an SMP/E USERMOD. The syntax and operands of the VSF2UOPT macro instructions are described in this section.

If you have Fortran applications that are link-edited with Language Environment, then there is no error option table to customize.

Use the VSF2UOPT macro to customize the Error Option Table as follows:

- Adding new error messages to the table, without changing existing ones, by coding the **VSF2UOPT Required Macro Instruction**, followed by an END statement.
- Changing existing error messages in the table, with or without adding new ones, by coding the **VSF2UOPT Required Macro Instruction**, followed by the necessary number of optional macro instructions, followed by an END statement.

For information about IBM-supplied error messages, see “Extended Error-Handling Subroutines and Error Option Table” in *VS FORTRAN Version 2 Language and Library Reference*.

Syntax of VSF2UOPT Required Macro Instruction

VSF2UOPT [ADDNTRY = *n*]

ADDNTRY=*n*

Is a positive integer specifying the number of new error message numbers to be added to the error option table. Additional error message numbers will begin at 500 and continue sequentially, up to a maximum of 899. If you want to change existing messages but do not want to add new ones, omit **ADDNTRY=*n***.

n Is a positive integer between 1 and 598.

Syntax of VSF2UOPT Optional Macro Instruction

VSF2UOPT MSGNO = (*erm sno[,qty]*)
[,ALLOW = *errs*]
[,INFOMSG = YES | NO]
[,IOERR = YES | NO]

[,MODENT = YES | NO]
[,PRINT = *prmsg*]
[,PRTBUF = YES | NO]
[,TRACBAK = YES | NO]
[,USREXIT = *exitname*]

The MSGNO option must always be specified. The default values of the five options INFOMSG, IOERR, MODENT, PRTBUF, and TRACBAK vary according to the following conditions:

- If the value of MSGNO specifies an IBM-supplied message number, and *none* of the five options is changed, then the default values are found in “Extended Error-Handling Subroutines and Error Option Table” of *VS FORTRAN Version 2 Language and Library Reference*.
- If either
 - The value of MSGNO specifies an IBM-supplied message number, and *one or more* of the five options is changed, or
 - The value of MSGNO specifies a new message number,

Then the default values for the unspecified options are the following:

INFOMSG
NO

IOERR
NO

MODENT
YES

PRTBUF
NO

TRACBAK
YES

MSGNO = (*ermsno* [, *qty*])

Specifies which error messages are affected by the default changes.

ermsno

Specifies either one message number, or the first error message number in a series of consecutive numbers.

qty

Specifies, if there is more than one, the number of consecutive error message numbers, beginning with *ermsno*.

For example, if the option is coded MSGNO=(153), then the default values for message 153 is changed. If the option is coded MSGNO=(153,4), then the default values for messages 153 through 156 is changed.

ALLOW = *errs*

Specifies the number of times the error can occur before the program is terminated.

errs

Specifies the number of errors allowed. To specify an exact number of errors allowed, *errs* must be a positive integer with a maximum of 255. A zero, or any number greater than 255, means that the error can occur an unlimited number of times.

Note: Altering an error option table entry to allow “unlimited” error occurrence might cause a program to loop indefinitely.

If the value of MSGNO specifies an IBM-supplied message number, the default value for this option is listed in “Extended Error-Handling Subroutines and Error Option Table” of *VS FORTRAN Version 2 Language and Library Reference* . If the value of MSGNO specifies a new message number, the default value is 10.

INFOMSG = YES | NO

Specifies whether the message is an informational or an error message.

YES

Specifies that the message is informational only. In this case the following occurs:

- No user error exit is taken.
- The value of ALLOW is ignored. Running will not terminate, even if it reaches the designated number of errors allowed.
- The error summary printed after termination of your program does not include a count of the number of times the condition occurred.

NO Specifies that the message is an error message.

IOERR = YES | NO

Specifies whether this error message represents an I/O error for which error counting is to be suppressed when an ERR or IOSTAT option is given on the I/O statement.

YES

Specifies that if an ERR or IOSTAT option is given, the occurrence of the error is not to be counted toward the maximum number specified by the ALLOW option above. This should be specified only for those errors listed in *VS FORTRAN Version 2 Language and Library Reference* for which the ERR and IOSTAT options are honored.

NO Specifies that the error occurrence is to be counted toward the maximum number of errors allowed.

MODENT = YES | NO

Specifies whether the ERRSET subroutine can be used to modify the error option table entry for this message.

YES

Specifies that the entry can be modified.

NO Specifies that the entry cannot be modified.

If you code a YES value for an IBM-supplied error message whose default is NO, and you then modify this entry using the ERRSET subroutine, you might receive undesirable results. Check the topic “Extended Error-Handling Subroutines and Error Option Table” of *VS FORTRAN Version 2 Language and Library Reference* to find out which message numbers have a “Modifiable Entry” value of NO.

PRINT = prmsg

Specifies the number of times the error message is to be printed. Subsequent occurrences of the error do not cause the message to be printed again.

prmsg

Specifies the number of times the message is to be printed. To specify an exact number of times printed, *prmsg* must be a positive integer, with a

maximum of 254. A "0" means the message will not be printed. Specifying 255 means the message can be printed an unlimited number of times.

If the value of MSGNO specifies an IBM-supplied message number, the default value for this option is listed in the chapter "Extended Error-Handling Subroutines and Error Option Table" in *VS FORTRAN Version 2 Language and Library Reference*. If the value of MSGNO specifies a new message number, the default value is 5.

PRTBUF = YES | NO

Specifies whether the I/O buffer is to be printed following certain I/O errors.

YES

Specifies that the contents of the buffer are to be printed.

NO Specifies that the contents of the buffer are not to be printed.

This option applies only to IBM-supplied error messages. Do not code YES unless the IBM-supplied default for this error message number already allows the buffer to be printed. Check the topic "Extended Error-Handling Subroutines and Error Option Table" in *VS FORTRAN Version 2 Language and Library Reference* to find out which message numbers have a "Print Buffer" value of YES.

TRACBAK = YES | NO

Specifies whether a module traceback listing is to be printed following the error message.

YES

Specifies that the traceback listing is to be printed.

NO Specifies that the traceback listing is not to be printed.

USREXIT = *exitname*

Specifies the user error exit routine that is invoked following the printing of the error message.

exitname

Specifies the entry point name of the user error exit routine. The routine should not be written in VS FORTRAN and should be reentrant.

If the routine is specified here, instead of being specified as an option passed to the ERRSET subroutine, the routine is invoked when the error occurs for any user. In this case, the routine will be invoked, regardless of whether the ERRSET routine was used or not. (However, unless a MODENT value of NO is in effect, programs can still call ERRSET dynamically to specify their own exit routine instead of the one specified by USREXIT.)

For programs operating in link mode, the user error exit routine must be link-edited with all users' programs.

To make the user error exit routine available to users who operate in load mode, the routine must be included in the composite module AFH5RENA. Then, if the user error exit routine must communicate with the program in which the error was detected, it must do so using a dynamic common area, not a static one.

Customizing Fortran LIBPACKS

The Fortran LIBPACKs are collections of individual modules that are packaged into a single load module in order to reduce the time that would otherwise be needed to load the individual modules.

Language Environment provides four Fortran LIBPACKs, which you can customize either during or following the installation of Language Environment.

Table 23. Fortran LIBPACKs

For applications link-edited with...	Customize LIBPACK...	Which is loaded...
Language Environment	AFHPRNAG	Above 16 MB
Language Environment	AFHPRNBG	Below 16 MB
VS FORTRAN	AFH5RENA	Above 16 MB
VS FORTRAN	AFH5RENB	Below 16 MB

The following tables give the names of the individual modules that can be included with or excluded from the LIBPACKs. In the tables, the terms **required** and **optional** are defined as follows:

required

Means that this module must be a part of the LIBPACK. It is not possible to exclude it.

optional

Means that this module may be either included or excluded from the LIBPACK. If the function indicated for the module is frequently used at your installation, the module should generally be included in order to avoid having to load it individually for each enclave.

For LIBPACKs loaded above the 16-MB line, the optional modules are included in the IBM-supplied default LIBPACK. For LIBPACKs loaded below the 16-MB line, only the required modules are included in the IBM-supplied default LIBPACK. Each optional LIBPACK module is also present individually. It will be loaded if that module is not included in the LIBPACK.

Refer to “Tailoring the Fortran LIBPACKs” on page 12 for information about how to tailor these LIBPACKs.

Contents of the Fortran LIBPACK AFHPRNAG

Table 24 lists routines you can include in the Fortran LIBPACK AFHPRNAG and briefly describes each to help you determine which to include in your tailored LIBPACK.

Note: For all entries in Table 24, the link-edited AMODE is 31 and the link-edited RMODE is ANY.

Table 24. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNAG

Name	Description	Required or Optional
AFHALBCG	Library common work area	Required
AFHBCITT	Character intrinsic functions	Optional
AFHBCMPT	Complex/character compare routine	Optional

Table 24. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNAG (continued)

Name	Description	Required or Optional
AFHBCMVT	Character move routine	Optional
AFHBCNCT	Character concatenation routine	Optional
AFHBCSTT	IBCLR/IBSET/BTEST functions	Optional
AFHBDPRT	Double/Extended precision product	Optional
AFHBFIFT	Real to integer intrinsic function	Optional
AFHBIBTT	IBITS using INTEGER*1 or INTEGER*2 argument	Optional
AFHBIDXT	Character index function	Optional
AFHBLOGT	Bit intrinsic functions, INTEGER*4 arguments	Optional
AFHBLXCT	Lexical comparison routines	Optional
AFHBMVBT	MVBITS (move bits) subroutine	Optional
AFHBMV8T	MVBITS (move bits) routine, INTEGER*8 arguments	Optional
AFHBMXDT	Maximum/minimum function, REAL*8 arguments	Optional
AFHBMXIT	Maximum/minimum function, INTEGER*4 arguments	Optional
AFHBMXRT	Maximum/minimum function, REAL*4 arguments	Optional
AFHBSHCT	ISHFTC function, all integer argument types	Optional
AFHBSHFT	ISHFT bit shift function, INTEGER*1 or INTEGER*2 arguments	Optional
AFHBMXST	Exponent underflow control function	Optional
AFHCBFBE	Condition token ownership	Optional
AFHCENAE	Fortran condition enablement	Required
AFHCGETT	Qualifying data retrieval function	Optional
AFHCLC1E	Locator text construction	Optional
AFHCLC2E	Message text construction	Optional
AFHCLOCT	Qualifying data address	Optional
AFHCLSHE	Language specific condition handler for math routines	Required
AFHCPUTT	Qualifying data update	Optional
AFHCQFBE	Feedback code query function	Optional
AFHCSERT	Compiler detected error processing at runtime	Optional
AFHCSGLE	Condition signaling processor	Required
AFHCTMHE	MTF termination condition handler	Optional
AFHCTOHE	I/O termination condition handler	Optional
AFHCTRAT	ERRTRA processing	Optional
AFHCXITE	Exit DSA activation	Optional
AFHDASGT	ASSIGNM (DCBS character) processor	Required
AFHDBGVE	DCBS given byte	Required
AFHDBMOE	DCBS assignment (move)	Required
AFHDBMVE	DCBS move string	Required
AFHDBPAE	DCBS pad string	Required
AFHDBTRE	DCBS truncate string	Required

Table 24. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNAG (continued)

Name	Description	Required or Optional
AFHDBTTE	DCBS translate and test	Required
AFHFGSTL	Math glue code generator	Optional
AFHGDIRE	Direct symbol table lookup	Optional
AFHGFORT	TEST option debug interface	Optional
AFHGISDE	Init symbol dictionary default	Optional
AFHGSQLE	Sequential lookup service	Optional
AFHIABDT	SYSABD processing	Optional
AFHIABNT	SYSABN processing	Optional
AFHIEINE	Enclave initialization	Required
AFHIETRE	Enclave termination	Required
AFHIEXTT	CALL EXIT processing	Optional
AFHIMTRT	Main program termination	Required
AFHIPAUT	PAUSE processing	Optional
AFHIPINE	Process initialization	Required
AFHIRCST	SYSRCS processing	Optional
AFHIRCTT	SYSRCT processing	Optional
AFHIRCXT	SYSRCX processing	Optional
AFHISTPT	STOP processing	Required
AFHITINE	Thread initialization	Required
AFHITTRE	Thread termination	Required
AFHLNABE	Find NAB and build dummy DSA	Required
AFHMOCBE	MTF runtime options for subtask	Required
AFHOASTE	Asynchronous I/O file close at termination routine	Optional
AFHOASYT	Asynchronous I/O request processing routine	Optional
AFHOBDSE	Build descriptor from parse tree	Optional
AFHOBNTE	Build nest table, implied DO in iolist item	Optional
AFHOBTRE	Build parse tree	Optional
AFHOCLOT	CLOSE processing routine	Optional
AFHOCMFE	I/O to terminal or to other device processing routine	Optional
AFHOCNTT	Control statement processing routine	Optional
AFHOCVIE	Copy parse tree or descriptor	Optional
AFHODCBE	DCB attributes resolution routine	Required
AFHODICT	DEFINE FILE processing routine	Optional
AFHODYNG	Dynamic file allocation	Optional
AFHOFINT	FILEINF processing routine	Optional
AFHOFMPE	Formatted I/O record processing routine	Optional
AFHOFMTT	Formatted I/O service request routing routine	Optional
AFHOFSCG	File name scan	Optional
AFHOIBCT	Pre-VS FORTRAN I/O services routing routine	Optional

Table 24. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNAG (continued)

Name	Description	Required or Optional
AFHOINIE	I/O support initialization	Required
AFHOINQT	INQUIRE statement processing routine	Optional
AFHOINTE	Internal file I/O service processing routine	Optional
AFHOLDFT	Pre-VSF 1.4.0 list-directed I/O parameter list processor	Optional
AFHOLDRT	List-directed I/O processing routine	Optional
AFHOLDTE	Pre-VSF 1.4.0 list-directed I/O processing routine	Optional
AFHONAMT	Pre-VSF 1.4.0 NAMELIST I/O parameter processor routine	Optional
AFHONLLE	Namelist I/O for static debug	Optional
AFHONLTE	Pre-VSF 1.4.0 NAMELIST I/O processing routine	Optional
AFHONMLT	Namelist I/O processing routine	Optional
AFHOOPNT	OPEN statement processing routine	Optional
AFHOSCOT	Pre-VSF 1.4.0 I/O services routing routine	Optional
AFHOSIIE	Get scalar intrinsic items	Optional
AFHOSTAG	Default I/O units allocation	Required
AFHOSYSE	STOP/PAUSE message display routine	Required
AFHOTRFE	Close all files at termination routine	Required
AFHOUFMT	Unformatted I/O processing routine	Optional
AFHOUFOE	Pre-VSF 1.4.0 unformatted I/O processing routine	Optional
AFHOUNIT	UNTANY/UNTNOFD processing	Optional
AFHOUTAG	Unit attribute table	Required
AFHPINIE	Program management initialization	Required
AFHPLVDE	LIBVEC descriptor	Required
AFHPRNAG	AFHPRNAG LIBPACK CSECT	Required
AFHRABTT	ABORT processing routine	Optional
AFHSDYAT	Obtain storage for ALLOCATE statement routine	Optional
AFHSDYDT	Free storage for DEALLOCATE statement routine	Optional
AFHSFREE	Storage free	Optional
AFHSGETE	Storage get	Optional
AFHSMIRE	Storage management initialization	Required
AFHSSG1T	Signal condition FOR0311S	Optional
AFHSSG2T	Signal condition FOR0312S	Optional
AFHSSG3T	Signal condition FOR0313S	Optional
AFHSVFAT	VSF version ALLOCATE/DEALLOCATE statements routine	Optional
AFHTCNIE	External input to internal format conversion routine	Optional
AFHTCNOE	Internal format to external output conversion routine	Optional
AFHTCVSE	I/O data conversion routing routine	Optional
AFHTCVTE	I/O data conversion routing routine adcon form	Optional
AFHTTENE	Powers of ten constants tables	Optional

Table 24. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNAG (continued)

Name	Description	Required or Optional
AFHUDMAE	Dump file attributes event handler	Optional
AFHUDM2E	Dump variable event handler	Optional
AFHUDUMT	Dump processing	Optional
AFHUSDMT	SDUMP processing	Optional
AFHVSPIT	Obtain compile-time required vector temporaries routine	Optional
AFHXARGT	Get argument string	Optional
AFHXBSDE	New direct symbol table lookup routine	Optional
AFHXCDME	Common block directory maintenance routine	Optional
AFHXCMNT	Obtain dynamic common blocks storage routine	Optional
AFHXCPTV	CPU time processing routine	Optional
AFHXCUIE	Compiled unit identification routine	Optional
AFHXCVD E	Convert and dump program symbols routine	Optional
AFHXDCLE	Save area classification routine	Optional
AFHXDEST	Signal extended common request routine	Optional
AFHXDIVT	DIV requests processing routine	Optional
AFHXDOCT	Divide check/overflow test routine	Optional
AFHXDPET	Signal parallel execution request routine	Optional
AFHXDSPT	Old form calculate array span/dimension factor routine	Optional
AFHXDTME	Termination exit to close DIV objects	Optional
AFHXDYLT	Dynamic loading processing routine	Optional
AFHXEINE	LCWA init for environment and runtime options	Required
AFHXEV7E	Fortran event handler routine	Required
AFHXFAIT	LCP initialize associated variable pointer routine	Optional
AFHXFAUT	LCP update associated variable routine	Optional
AFHXFFEE	Identify entry point type routine	Optional
AFHXFMTT	LCP define file processing routine	Optional
AFHXIGNT	IGNORE FILE HISTORY processing routine	Optional
AFHXLNKT	Nonshareable to shareable CSECT linkage routine	Optional
AFHXOWNE	Save area ownership routine	Optional
AFHXPMLT	Subprogram parameter list checker routine	Optional
AFHXSIDE	Obtain ISN or sequence number id routine	Optional
AFHXSISE	Convert item to vib_desc_fmt	Optional
AFHXSPNT	Calculate array span/dimension factor routine	Optional
AFHXSQLE	New sequential symbol table retrieval	Optional
AFHXSTIE	Obtain symbol table information routine	Optional
AFHXTIMT	Date/time information routine	Optional
AFHXUSDE	Update symbol table retrieval	Optional
AFHX8SMT	New compiler i*8 simulator routine	Optional

Contents of the Fortran LIBPACK AFHPRNBG

Table 25 lists routines you can include in the Fortran LIBPACK AFHPRNBG and briefly describes each to help you determine which to include in your tailored LIBPACK.

Note: For all entries in Table 25, the link-edited AMODE is ANY and the link-edited RMODE is 24.

Table 25. Routines eligible for inclusion in the Fortran LIBPACK AFHPRNBG

Name	Description	Required or Optional
AFHLCLNE	Clear Fortran dummy DSA	Required
AFHOASUG	Asynchronous I/O subtask routine	Optional
AFHOBDRE	Direct I/O processing routine	Optional
AFHOBSQE	Sequential I/O processing routine	Required
AFHOFSTG	File status	Required
AFHOSTRE	Striped I/O processing routine	Optional
AFHOVKYE	VSAM KSDS (keyed I/O) services routine	Optional
AFHOVSMG	VSAM (RRDS, ESDS) I/O services routine	Optional
AFHPRNBG	AFHPRNBG LIBPACK CSECT	Required

Contents of the Fortran LIBPACK AFH5RENA

Table 26 lists routines you can include in the Fortran LIBPACK AFH5RENA and briefly describes each to help you determine which to include in your tailored LIBPACK.

Note: For all entries in Table 26, the link-edited AMODE is 31 and the link-edited RMODE is ANY.

Table 26. Routines eligible for inclusion in the Fortran LIBPACK AFH5RENA

Name	Description	Required or Optional
AFH5ABEX	VSF ABEND handler (ESTAE)	Required
AFH5ALOP	VAL function routine	Optional
AFH5AMEP	VSF NAMELIST I/O parmlist decoder	Optional
AFH5AREN	VSF VRENA vector table	Required
AFH5ARGP	VSF 2.6 ARG obtain argument string routine	Optional
AFH5ASYP	Asynchronous I/O services driver routine	Optional
AFH5BALG	Vector boundary alignment routine	Optional
AFH5BCOP	Old FORTRAN library services interface routine	Optional
AFH5BLN\$	VSF build nest table stub	Required
AFH5BLNT	Build nest table I/O service routine	Optional
AFH5CDM\$	VSF dynamic COMMON routine special stub	Required
AFH5CDMA	VSF COMMON block directory maintenance	Optional
AFH5CLOP	VSF CLOSE services routine	Optional
AFH5CNI\$	VSF conversion routine special stub	Required

Table 26. Routines eligible for inclusion in the Fortran LIBPACK AFH5RENA (continued)

Name	Description	Required or Optional
AFH5CNO\$	VSF conversion routine special stub	Required
AFH5COM\$	VSF COMH special stub	Required
AFH5COMH	VSF formatted I/O processor	Optional
AFH5CONI	VSF convert external to internal format	Optional
AFH5CONO	VSF convert internal to external format	Optional
AFH5CPTP	VSF CPU TIME routine	Optional
AFH5CVT\$	VSF CVTH special stub	Required
AFH5CVTH	VSF conversion routine	Optional
AFH5DEB\$	VSF DEBU special stub	Required
AFH5DFCP	VSF DEFINEFILE processing routine	Optional
AFH5DFIP	VSF pre-1.4.0 list-directed I/O decoder	Optional
AFH5DIO\$	VSF DIOS special stub	Required
AFH5DIVP	VSF Data-In-Virtual services processor	Optional
AFH5DOCP	VSF divide check/overflow test routine	Optional
AFH5DYLP	VSF dynamic binder routine	Optional
AFH5DYN\$	VSF dynamic allocation special stub	Required
AFH5DYNA	VSF dynamic file allocation routine	Optional
AFH5EMG\$	VSF error message special stub	Required
AFH5EMGN	VSF message build routine	Optional
AFH5ERE\$	VSF EEH special stub	Required
AFH5ERRE	VSF object time error summary	Required
AFH5ERS\$	VSF exit/return code special stub	Required
AFH5EXIP	VSF return code and exiting routine	Optional
AFH5FINP	VSF file information routine	Optional
AFH5FISC	VSF file name scan routine	Optional
AFH5FNTH	VSF program interrupt handler	Required
AFH5GMFM	VSF getmain/freemain routine	Required
AFH5GPRM	VSF global parmlist	Required
AFH5IAD\$	VSF IAD interface special stub	Required
AFH5IIO\$	VSF internal I/O special stub	Required
AFH5IIO\$	VSF internal I/O routine	Optional
AFH5INI\$	VSF Vector common init special stub	Required
AFH5INQP	VSF INQUIRE processing routine	Optional
AFH5INTH	VSF vector program interrupt handler	Optional
AFH5INTP	VSF init/term routine	Required
AFH5IOCP	VSF I/O control processing	Optional
AFH5IOFP	VSF formatted I/O router routine	Optional
AFH5IOLP	VSF list-directed processor	Optional
AFH5IONP	VSF NAMELIST processor	Optional

Table 26. Routines eligible for inclusion in the Fortran LIBPACK AFH5RENA (continued)

Name	Description	Required or Optional
AFH5IOUP	VSF unformatted I/O processor	Optional
AFH5KIO\$	VSF keyed I/O special stub	Required
AFH5LBC0	VSF library common work area	Required
AFH5LINP	VSF shareable code load routine	Optional
AFH5LOAD	VSF load/delete service routine	Required
AFH5LOC\$	VSF offset locate special stub	Required
AFH5LOCA	VSF offset locator routine	Optional
AFH5MIN\$	VSF MTF init special stub	Required
AFH5MMA\$	VSF MTF map and attach special stub	Required
AFH5MOPP	VSF extended error handling routine	Optional
AFH5MPR\$	MTF subparameter parser special stub	Required
AFH5MSKL	VSF message skeletons	Optional
AFH5OCMP	VSF dynamic COMMON processor routine	Optional
AFH5OPEP	VSF OPEN processor routine	Optional
AFH5PARM	VSF runtime parameter list scan routine	Required
AFH5PIO\$	VSF striped I/O special stub	Required
AFH5POS\$	VSF post-ABEND processor special stub	Required
AFH5RDCB	VSF DCB resolution routine	Required
AFH5SCOP	VSF pre-1.4 I/O interface	Optional
AFHFSPAP	VSF array span calculator	Optional
AFH5SPBP	VSF 1.4 array span calculator	Optional
AFH5SPIE	VSF SPIE set routine	Required
AFH5STAE	STAE set routine	Required
AFH5STIO	VSF standard I/O setup routine	Required
AFH5TIMP	VSF obtain date and time routine	Optional
AFH5TRC\$	VSF traceback special stub	Required
AFH5TRCH	VSF traceback routine	Optional
AFH5TRMF	VSF termination file close routine	Required
AFH5UNIN	VSF vector unnorm argument exception handler	Optional
AFH5UOPT	VSF error message options table	Required
AFH5VDMQ	VSF PDUMP/CPDUMP service routine	Optional
AFH5VINI	VSF vector common area initializer	Optional
AFH5VIO\$	VSF non-keyed VSAM special stub	Required
AFH5VTEN	VSF floating point conversion constants	Optional
AFH5VUAT	VSF UNIT Attribute Table	Required

Contents of the Fortran LIBPACK AFH5RENB

Table 27 on page 246 lists routines you can include in the Fortran LIBPACK AFH5RENB and briefly describes each to help you determine which to include in your tailored LIBPACK.

Note: For all entries in Table 27, the link-edited AMODE is ANY and the link-edited RMODE is 24.

Table 27. routines eligible for inclusion in the Fortran LIBPACK AFH5RENB

Name	Description	Required or Optional
AFH5ASUB	Asynchronous I/O services subtask routine	Optional
AFH5BREN	VSF VRENB locator table	Required
AFH5DIOS	VSF direct access I/O routine	Optional
AFH5FIST	VSF file info status routine	Required
AFH5KIOS	VSF keyed I/O processor	Optional
AFH5SIOS	VSF sequential I/O routine	Required
AFH5VIOS	VSF non-keyed VSAM routine	Optional
IBMPEV11 CEEEV011	Enterprise PL/I library	ANY

Appendix C. Modules eligible for the link pack area

The modules listed in the following table can be put in the LPA or the ELPA, depending on their RMODE:

- If the RMODE is ANY, the module can reside in the link pack area or in the extended link pack area (above or below the 16-MB line).
- If the RMODE is 24, the module can reside only in the link pack area (below the 16-MB line).

If you are considering placing the modules listed in this in the LPA or the ELPA, IBM highly suggests that you place the SCEELPA data set in the LPA list (LPALSTxx). This data set contains modules that are reentrant, reside above the line and are heavily used by z/OS itself.

The specific HLL sections contains tables of modules eligible for the LPA or the ELPA above and beyond what is found in the SCEELPA data set. You will need to use the Dynamic LPA or MLPA approach to move these modules into the LPA/ELPA. You do not need to include recommended modules if they contain functions your installation does not use. Language Environment modules not listed in these tables can be moved into LPA/ELPA at your discretion.

Language Environment base modules

Modules and *aliases* listed in Table 28 can be added into LPA/ELPA by using the sample job CEEWLPA that is found in the SCEESAMP data set.

Table 28. Language Environment modules eligible for inclusion in the link pack area and the extended link pack area

Language Environment module name	Description	RMODE
CEEBINIT CEEBLIBM	Initialization/termination for batch	24
CEEBLIA IBMBLIA IBMBPIIA	OS PL/I and C load module compatibility	24
CEEBLRR	Library Retention Routine	ANY
CEEBPICI	Initialization/termination routines for preinitialization compatibility	24
CEELRRIN	LRR initialization	ANY
CEELRRXP	LRR Initialization that permits XPLINK	ANY
CEELRRTR	LRR termination	ANY
CEEMENU0	Message file with mixed-case English; messages 000-999	ANY
CEEMENU2	Message file with mixed-case English; messages 2000-2999	ANY
CEEMENU3	Message file with mixed-case English; messages 3000-3999	ANY
CEEMENU4	Message file with mixed-case English; messages 4000-4999	ANY
CEEMENU5	Message file with mixed-case English; messages 5000-5999	ANY
CEEMJPN0	Message file with Japanese; messages 000-999	ANY

Table 28. Language Environment modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Language Environment module name	Description	RMODE
CEEMJPN2	Message file with Japanese; messages 2000-2999	ANY
CEEMJPN3	Message file with Japanese; messages 3000-3999	ANY
CEEMJPN4	Message file with Japanese; messages 4000-4999	ANY
CEEMJPN5	Message file with Japanese; messages 5000-5999	ANY
CEEMUEN0	Message file with uppercase English; messages 000-999	ANY
CEEMUEN2	Message file with uppercase English; messages 2000-2999	ANY
CEEMUEN3	Message file with uppercase English; messages 3000-3999	ANY
CEEMUEN4	Message file with uppercase English; messages 4000-4999	ANY
CEEMUEN5	Message file with uppercase English; messages 5000-5999	ANY
CEEPIPI	Initialization/termination routines for the Language Environment preinitialization facility	24

Note: Modules added to LPA must also remain in SCEERUN.

Language Environment C/C++ component modules

The C/C++ component modules and *aliases* listed in Table 29 can be moved into LPA/ELPA using the sample job EDCWLPA found in the SCEESAMP data set. The Language Environment base modules listed in Table 28 on page 247 should also be moved into LPA/ELPA.

Table 29. C/C++ modules eligible for inclusion in the link pack area and the extended link pack area

C/C++ module name	Description	RMODE
	Demangler	ANY
EDCHDMNP DEMANGLE		
EDCNINSP	Debug tool interface	ANY
EDCPRLK	Prelink utility	ANY
	Rename list	ANY
EDCRNLB EDCRNLST		
EDCZEMSG	Mixed-case US English messages	ANY
EDCZJMSG	Japanese messages	ANY
EDCZUMSG	Uppercase English messages	ANY
IEDCMSGT	C/370 message table	ANY
	DLL for XPLINK C++ applications	ANY
CELHDCPP SCEECPP		
CELVH003	C++ runtime— for AMODE 31 XPLINK applications	ANY

Note:

1. EDCNINSP is highly recommended for inclusion in the LPA or ELPA if the Debug tool is heavily used.
2. EDCPRLK is highly recommended for inclusion in the LPA or ELPA if the prelink utility is heavily used.
3. The default code page converters or locale modules, or customized code page converters or locale modules (the ones applicable for the user's country), should be included in the LPA or ELPA.

Language Environment COBOL component modules

The COBOL component modules and *aliases* listed in Table 30 can be moved into LPA/ELPA using the sample job IGZWMLP4 found in the SCEESAMP data set. The Language Environment base modules listed in Table 28 on page 247 should also be moved into LPA/ELPA.

Additional modules that exist for OS/VS COBOL compatibility (ILBO) are not described here. Refer to the OS/VS COBOL documentation for information about these modules.

Table 30. COBOL modules eligible for inclusion in the link pack area and the extended link pack area

COBOL module name	Description	RMODE
CDAEED	COBOL Member 4 utility routines	CDA API library
CEEEV004	COBOL Member 4 event handler	ANY
CEEEV005	COBOL event handler	ANY
IGZCA2D	DBCS data manipulation	ANY
IGZCD2A	DBCS data manipulation	ANY
IGZCMTUE	COBOL WTO error messages	ANY
IGZCPAC	COBPACK	ANY
IGZCPCO	COBPACK	ANY
IGZINSH	Formatted dump and Debug Tool support	ANY
IGZEPL	COBOL termination (VS COBOL II and OS/VS COBOL only)	24
IGZERRE	COBOL reusable environment	ANY
IGZEWTO	COBOL: write error message to operator's console	ANY
IGZCWTO	COBOL write error message	ANY
IGZCD24	COBOL dynamic call to AMODE(24) programs	24
IGZCMGUE	COBOL (IGZ) messages in uppercase English	ANY
IGZCMGEN	COBOL (IGZ) messages in English	ANY
IGZCMGJA	COBOL (IGZ) messages in Japanese	ANY
IGZCMLT IIGZMSGT	COBOL message tables	ANY
IGZEPLF	COBOL environment initialization (VS COBOL II and OS/VS COBOL only)	24
IGZCBUG	Used for debugging	24
IGZCLNC	Linkage manager for OS/VS COBOL and IGZBRDGE (dynamic call and cancel)	24

Table 30. COBOL modules eligible for inclusion in the link pack area and the extended link pack area (continued)

COBOL module name	Description	RMODE
IGZCLNK	Linkage manager for VS COBOL II and COBOL/370 (dynamic call and cancel)	24
IGZCULE	User I/O logic error handler	24
IGZCXFR	I/O declarative transfer	24
IGZEDMR	Reusable environment deactivation	24
IGZEINI	Environment initialization	24
IGZEINP	Accept input reader	24
IGZEOPN	OPENS SYSIN and SYSPUNCH in the Initial Program Thread (IPT)	24
IGZEOUT	Display output writer	24
IGZEQBL	QSAM initialization transmission verbs, error exits	24
IGZEQOC	QSAM OPEN/CLOSE	24
IGZERCO	OS/VS COBOL TERMINATION	24
IGZESMG	Sort/Merge interface	24
IGZEVAM	VSAM-to-IDCAMS interface	24
IGZEVEX	VSAM exit module for SYNAD and LERAD	24
IGZESCD	SORT-CONTROL I/O handling routine	24
IGZETRM	Environment termination	24
IGZLLIBV	COBOL Member 4 library vector	ANY
IGZXAPI	Runtime information query	ANY
IGZDMR	Reusable information support	24
IGZXD24	Dynamic call manager	24
IGZXLPIO	I/O manager	ANY
IGZXLPKA	COBOL Member 4 library services 1	ANY
IGZXLPKB	COBOL Member 4 library services 2	24
IGZXLPKC	COBOL Member 4 core library	ANY
IGZXP2	COBOL Member 4 utility routines	ANY

Language Environment Fortran component modules

The Fortran component modules and *aliases* listed in Table 31 on page 251 can be moved into LPA/ELPA using the sample job AFHWMLP2 found in the SCEESAMP data set. The Language Environment base modules listed in Table 28 on page 247 should also be moved into LPA/ELPA.

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area

Fortran module name	Description	RMODE
AFHBCITT AFHBACHK AFHBACIK AFHBACJK AFHBACKK AFHBCHAR AFHBCH2R AFHBCH8R AFHBIACK AFHBICHR AFHBJACK AFHBJCHR AFHBLNR AFHBLN8R	Character intrinsic functions	ANY
AFHBCMPT AFHBCMPR AFHBXMPR	Complex/character compare routine	ANY
AFHBCMVT AFHBCMVR	Character move routine	ANY
AFHBCNCT AFH-BCNCK AFHBCNCR	Character concatenation routine	ANY
AFHBCSTT AFHBHCLK AFHBHCLR AFHBHSTK AFHBHSTR AFHBHTSK AFHBHTSR AFHBKCLK AFHBKCLR AFHBKSTK AFHBKSTR AFHBKTSK AFHBKTSR	IBCLR/IBSET/BTEST functions	ANY
AFHBDPRT AFHBDPRR AFHBQPRR	Double/Extended precision product	ANY
AFHBFIFT AFHBIDTR AFHBIFIR AFHBINTR	Real to integer intrinsic function	ANY
AFHBIBTT AFHBHBTK AFHBHBTR AFHBKBTk AFHBKBTR	IBITS using INTEGER*1 or INTEGER*2 argument	ANY
AFHBIDXT AFHBIDXK AFHBIDXR AFHBJDXK AFHBJDXR	Character index function	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHBSHCT AFHBISCK AFHBISCR AFHBJSCK AFHBJSCR AFHBKSK AFHBKSCR AFHBHCK AFHBHSCR	ISHFTC function, all integer argument types	ANY
AFHBSHFT AFHBHLSK AFHBHLR AFHBHRSK AFHBHRSR AFHBHSHK AFHBHSHR AFHBKLSK AFHBKLSR AFHBKRSK AFHBKRSR AFHBKSHK AFHBKSHR	ISHFT bit shift function, INTEGER*1 or INTEGER*2 arguments	ANY
AFHBXMST AFHBXMSR	Exponent underflow control function	ANY
AFHCBFBE AFHCBFBR	Condition token ownership	ANY
AFHCGETT AFHCGETR	Qualifying data retrieval function	ANY
AFHCLC1E AFHCLC1R	Locator text construction	ANY
AFHCLC2E AFHCLC2R	Message text construction	ANY
AFHCMSGE AFHCMSGR IFORMSGT	Fortran message table header	ANY
AFHCMS1E AFHCMS1R	Mixed-case English message file 1	ANY
AFHCMS1J	Japanese message file 1	ANY
AFHCMS1U	Uppercase English message file 1	ANY
AFHCMS2E AFHCMS2R	Mixed-case English message file 2	ANY
AFHCMS2J	Japanese message file 2	ANY
AFHCMS2U	Uppercase English message file 2	ANY
AFHCMS3E AFHCMS3R	Mixed-case English message file 3	ANY
AFHCMS3J	Japanese message file 3	ANY
AFHCMS3U	Uppercase English message file 3	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHCMS4E AFHCMS4R	Mixed-case English message file 4	ANY
AFHCMS4J	Japanese message file 4	ANY
AFHCMS4U	Uppercase English message file 4	ANY
AFHCPUTT AFHCPUTR	Qualifying data update	ANY
AFHCQFBE AFHCQFBR	Feedback code query function	ANY
AFHCSSERT AFHCSEERR	Compiler detected error processing at runtime	ANY
AFHCTMHE AFHCTMHR	MTF termination condition handler	ANY
AFHCTOHE AFHCTOHR	I/O termination condition handler	ANY
AFHCTRAT AFHCTRAR	ERRTRA processing	ANY
AFHCXITE AFHCXITR	Exit DSA activation	ANY
AFHFGSTL AFHFGSTR	Math glue code generator	ANY
AFHGDIRE AFHGDIRR	Direct symbol table lookup	ANY
AFHGFORT AFHGSTNR AFHGSTXR AFHGTRCR	TEST option debug interface	ANY
AFHGISDE AFHGISDR	Init symbol dictionary default	ANY
AFHGSQLE AFHGSQLR	Sequential lookup service	ANY
AFHIABDT AFHIABDR	SYSABD processing	ANY
AFHIABNT AFHIABNR	SYSABN processing	ANY
AFHIEXTT AFHIEXTR	CALL EXIT processing	ANY
AFHIPAUT AFHIPAUK AHHIPAUT	PAUSE processing	ANY
AFHIRCST AFHIRCSR	SYSRCS processing	ANY
AFHIRCTT AFHIRCTR	SYSRCT processing	ANY
AFHIRCXT AFHIRCXR	SYSRCX processing	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHMMMAAG AFHMMMAAR	MTF map and ATTACH routine	24
AFHMSTCT AFHMSTCR	MTF subtask control	24
AFHMTFAG ³	MTF LIBPACK	ANY
AFHOASTE AFHOASTR	Asynchronous I/O file close at termination routine	ANY
AFHOASUG AFHOASUR	Asynchronous I/O subtask routine	24
AFHOASYT AFHOAINR AFHOAOUR AFHOAWTR	Asynchronous I/O request processing routine	ANY
AFHOBDRE AFHOBDRR	Direct I/O processing routine	24
AFHOBDSE AFHOBDSR	Build descriptor from parse tree	ANY
AFHOBNTE AFHOBNTR	Build nest table, implied DO in iolist item	ANY
AFHOBTRE AFHOBTRR	Build parse tree	ANY
AFHOCLOT AFHOCLOR	CLOSE processing routine	ANY
AFHOCMFE AFHOCMFR	I/O to terminal or to other device processing routine	ANY
AFHOCNTT AFHOCBSR AFHOCDLR AFHOCEFR AFHO CRWR	Control statement processing routine	ANY
AFHOCVIE AFHOCVIR	Copy parse tree or descriptor	ANY
AFHODICT AFHODICR	DEFINE FILE processing routine	ANY
AFHODYNG AFHODYNR	Dynamic file allocation	ANY
AFHOFINT AFHOFINR	FILEINF processing	ANY
AFHOFMPE AFHOFMPR	Formatted I/O record processing routine	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE		
AFHOFMTT	Formatted I/O service request routing routine	ANY		
AFHOCSFR				
AFHODSFR				
AFHOESFR				
AFHOFXFR				
AFHOIXFR				
AFHOQKFR				
AFHORDFR				
AFHORIFR				
AFHORKFR				
AFHORSFR				
AFHOSXFR				
AFHOUVFR				
AFHOWDFR				
AFHOWIFR				
AFHOWKFR				
AFHOWSFR				
AFHOFSCG			File name scan	ANY
AFHOFSCR				
AFHOIBCT	Pre-VS FORTRAN I/O services routing routine	ANY		
AFHOIAFR				
AFHOIANR				
AFHOIBSR				
AFHOIEFR				
AFHOIENR				
AFHOILFR				
AFHOILNR				
AFHOINFR				
AFHOIPAR				
AFHOIRFR				
AFHOIRNR				
AFHOIRWR				
AFHOIWFR				
AFHOIWRN				
AFHOINQT			INQUIRE statement processing routine	ANY
AFHOINQR				
AFHOINTE			Internal file I/O service processing routine	ANY
AFHOINTR				
AFHOLDFT	Pre-VSF 1.4.0 list-directed I/O parameter list processor	ANY		
AFHOLFAR				
AFHOLFER				
AFHOLFLR				
AFHOLFRR				
AFHOLFWR				
AFHOLVAR				
AFHOLVER				
AFHOLVLR				
AFHOLVRR				
AFHOLVWR				

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHOLDRT AFHOCSLR AFHODSLR AFHOESLR AFHOFXLR AFHOIXLR AFHORILR AFHORSLR AFHOWILR AFHOWSLR	List-directed I/O processing routine	ANY
AFHOLDTE AFHOAXLR AFHOLXLR AFHOMXLR AFHOTXLR	Pre-VSF 1.4.0 list-directed I/O processing routine	ANY
AFHONAMT AFHONFRR AFHONFWR AFHONVRR AFHONVWR	Pre-VSF 1.4.0 NAMELIST I/O parameter processing routine	ANY
AFHONLLE AFHONLWR	Namelist I/O for static debug	ANY
AFHONLTE AFHOSSNR AFHOXSNR	Pre-VSF 1.4.0 NAMELIST I/O processing routine	ANY
AFHONMLT AFHOCSNR AFHOESNR AFHORINR AFHORSNR AFHOWINR AFHOWSNR	Namelist I/O processing routine	ANY
AFHOSCOT AFHOVAFR AFHOVANR AFHOVBKR AFHOVEFR AFHOVENR AFHOVLFR AFHOVLNR AFHOVNFR AFHOVRFR AFHOVRNR AFHOVRWR AFHOVWFR AFHOVWNR	Pre-VSF 1.4.0 I/O services routing routine	ANY
AFHOSIIE AFHOSIIR	Get scalar intrinsic items	ANY
AFHOSTRE AFHOSTRR	Striped I/O processing routine	24

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHOUFMT AFHOFDUR AFHOFXUR AFHOIXUR AFHOQKUR AFHORDUR AFHORKUR AFHORSUR AFHOSXUR AFHOUVUR AFHOWDUR AFHOWKUR AFHOWSUR	Unformatted I/O processing routine	ANY
AFHOUFOE AFHOEXUR AFHOLXUR AFHOMXUR AFHOPXUR	Pre-VSF 1.4.0 unformatted I/O processing routine	ANY
AFHOUNIT AFHOUNFR AFHOUNTR	UNTANY/UNTNOFD processing	ANY
AFHOVKYE AFHOVKYR	VSAM KSDS (keyed I/O) services routine	24
AFHOVSMG AFHOVDIR AFHOVSQR	VSAM (RRDS, ESDS) I/O services routine	24
AFHPRNAG ¹ CEEEV007	AFHPRNAG LIBPACK CSECT	ANY
AFHPRNBG ¹	AFHPRNBG LIBPACK CSECT	24
AFHRABTT AFHRABTK	ABORT processing routine	ANY
AFHSDYAT AFHSDYAR	Obtain storage for ALLOCATE statement routine	ANY
AFHSDYDT AFHSDYDR AFHSDYFK	Free storage for DEALLOCATE statement routine	ANY
AFHSFREE AFHSFRER	Storage free	ANY
AFHSGETE AFHSGETR	Storage get	ANY
AFHSSG1T AFHSSG1R	Signal condition FOR0311S	ANY
AFHSSG2T AFHSSG2R	Signal condition FOR0312S	ANY
AFHSSG3T AFHSSG3R	Signal condition FOR0313S	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHRSVFAT AFHRSVALK AFHRSVALR AFHRSVA4K AFHRSVA4R AFHRSVA8K AFHRSVA8R AFHRSVDEK AFHRSVDER	VSF version ALLOCATE/DEALLOCATE statements routine	ANY
AFHTCVSE AFHTFAOR AFHTFCOR AFHTFDOR AFHTFEOR AFHTFGOR AFHTFIOR AFHTFLOR AFHTFQOR AFHTFZOR	I/O data conversion routing routine	ANY
AFHTCVTE AFHTCVTR	I/O data conversion routing routine adcon form	ANY
AFHUDMAE AFHUDMAR	Dump file attributes event handler	ANY
AFHUDM2E AFHUDM2R	Dump variable event handler	ANY
AFHUDUMT AFHUCDMR AFHUCPDR AFHUDUMR	Dump processing	ANY
AFHUSDMT AFHUSDMR	SDUMP processing	ANY
AFHVSPLIT AFHVSPTR	Obtain compile-time required vector temporaries routine	ANY
AFHXARGT AFHXARGR	Get argument string	ANY
AFHXBSDE AFHXBSDR	New direct symbol table lookup routine	ANY
AFHXCDME AFHXCDMR	Common block directory maintenance routine	ANY
AFHXCMNT AFHXCMNR AFHXCMSR AFHXDCDR AFHXDCFR AFHXDCGR AFHXDCIR AFHXSDCR	Obtain dynamic common blocks storage routine	ANY
AFHXCPTV AFHXCPTR	CPU time processing routine	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHXCUIE AFHXCUIR	Compiled unit identification routine	ANY
AFHXCVIDE AFHXCVIDR	Convert and dump program symbols routine	ANY
AFHXDCLE AFHXDCLR	Save area classification routine	ANY
AFHXDEST AFHXDESR	Signal extended common request routine	ANY
AFHXDIVT AFHXDCMR AFHXDNFR AFHXDNVR AFHXDRSR AFHXDSVR AFHXDTFR AFHXDTVR AFHXDWFR AFHXDWVR	DIV requests processing routine	ANY
AFHXDOCT AFHXDVKR AFHXOVER	Divide check/overflow test routine	ANY
AFHXDPET AFHXDPER	Signal parallel execution request routine	ANY
AFHXDSPT AFHXDSNR AFHXDS2R	Old form calculate array span&slash dimension factor routine	ANY
AFHXDTME AFHXDTMR	Termination exit to close DIV objects	ANY
AFHXDYLT AFHXDYLK AFHXDYLR	Dynamic loading processing routine	ANY
AFHXFAIT AFHXFAIR	LCP initialize associated variable pointer routine	ANY
AFHXFAUT AFHXFAUR	LCP update associated variable routine	ANY
AFHXFFEE AFHXFFER	Identify entry point type routine	ANY
AFHXFMTT AFHXFMTR	LCP define file processing routine	ANY
AFHXIGNT AFHXIGDR AFHXIGUR	IGNORE FILE HISTORY processing routine	ANY
AFHXLNKT AFHXLIMK AFHXLIMR AFHXLISK AFHXLISR	Nonshareable to shareable CSECT linkage routine	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFHXOWNE AFHXOWNR	Save area ownership routine	ANY
AFHXPMLT AFHXPMLK AFHXPMLR AFHXPPMK	Subprogram parameter list checker routine	ANY
AFHXSIDE AFHXSIDR	Obtain ISN or sequence number id routine	ANY
AFHXSISE AFHXSISR	Convert item to vib_desc_fmt	ANY
AFHXSPNT AFHXSP4R AFHXSP5R	Calculate array span/dimension factor routine	ANY
AFHXSQLE AFHXSQLR	New sequential symbol table retrieval	ANY
AFHXSTIE AFHXSTIR	Obtain symbol table information routine	ANY
AFHXTIMT AFHXCLKR AFHXCLXR AFHXDMTR AFHXDTXR	Date/time information routine	ANY
AFHXUSDE AFHXUSDR	Update symbol table retrieval	ANY
AFH5ALOP AFBVALOP	VAL function routine	ANY
AFH5AMEP AFBNAMEP IFYNAMEP	VSF NAMELIST I/O parmlist decoder	ANY
AFH5ARGP AFBVARGP	VSF 2.6 ARG obtain argument string routine	ANY
AFH5ASUB AFBVASUB	Asynchronous I/O services subtask routine	24
AFH5ASYP AFBVASYP IFYVASYP	Asynchronous I/O services driver routine	ANY
AFH5BALG AFVBALG	Vector boundary alignment routine	ANY
AFH5BCOP AFBIBCOPIFYBCOP	Old FORTRAN library services interface routine	ANY
AFH5BLNT AFBVBLNT IFYVBLNT	Build nest table I/O service routine	ANY
AFH5CDMA AFBVCDMA	VSF COMMON block directory maintenance	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFH5CLOP AFBVCLOP IFYVCLOP	VSF CLOSE services routine	ANY
AFH5COMH AFBVCOMH	VSF formatted I/O processor	ANY
AFH5CONI AFBVCONI	VSF convert external to internal format	ANY
AFH5CONO AFBVCONO	VSF convert internal to external format	ANY
AFH5CPTP AFBCCPTP AFBVCPTP	VSF CPUTIME routine	ANY
AFH5CVTH AFBVCVTH	VSF conversion routine	ANY
AFH5DFCP AFBDIOCP IFYDIOCP	VSF DEFINEFILE processing routine	ANY
AFH5DFIP AFBLDFIP IFYLDFIP	VSF pre-1.4.0 list-directed I/O decoder	ANY
AFH5DIOS AFBVDIOS	VSF direct access I/O routine	24
AFH5DIVP AFBVDIVP	VSF Data-In-Virtual services processor	ANY
AFH5DOCP AFBVDOCP	VSF divide check/overflow test routine	ANY
AFH5DYLP AFBVDYLP	VSF dynamic binder routine	ANY
AFH5DYNA AFBCDYNA AFBVDYNA	VSF dynamic file allocation routine	ANY
AFH5EMGN AFBVEMGN	VSF message build routine	ANY
AFH5EXIP AFBVEXIP	VSF return code and exiting routine	ANY
AFH5FINP AFBVFINP	VSF file information routine	ANY
AFH5FISC AFBCFISC AFBVFISC	VSF file name scan routine	ANY
AFH5IIOS AFBVIIOS	VSF internal I/O routine	ANY
AFH5INQP AFBVINQP IFYVINQP	VSF INQUIRE processing routine	ANY
AFH5INTH AFBVINTH	VSF vector program interrupt handler	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFH5IOCP AFBVIOCP IFYVIOCP	VSF I/O control processing	ANY
AFH5IOFP AFBVIOFP IFYVIOFP	VSF formatted I/O router routine	ANY
AFH5IOLP AFBVOLP IFYVIOLP	VSF list-directed processor	ANY
AFH5IONP AFBVIONP IFYVIONP	VSF NAMELIST processor	ANY
AFH5IOUP AFBVIOUP IFYVIOUP	VSF unformatted I/O processor	ANY
AFH5KIOS AFBVKIOS	VSF keyed I/O processor	24
AFH5LINP AFBVLINP IFYVLINP	VSF shareable code load routine	ANY
AFH5LOCA AFBVLOCA	VSF offset locator routine	ANY
AFH5MOPP AFBVMOPP IFYVMOPP	VSF extended error handling routine	ANY
AFH5MSKL AFBVMSKL	VSF message skeletons	ANY
AFH5OCMP AFBDDCMP AFBVOCMP IFYDDCMP	VSF dynamic COMMON processor routine	ANY
AFH5OPEP AFBVOPEP IFYVOPEP	VSF OPEN processor routine	ANY
AFH5RENA ¹ AFBVRENA	AFH5RENA LIBPACK CSECT	ANY
AFH5RENB ¹ AFBVRENB	AFH5RENB LIBPACK CSECT	24
AFH5RENP ¹ AFBVRENP	AFH5RENP LIBPACK CSECT	ANY
AFH5SCOP AFBVSCOP IFYVSCOP	VSF pre-1.4 I/O interface	ANY
AFH5SPAP AFBVSPAP IFYVSPAP	VSF array span calculator	ANY

Table 31. Fortran modules eligible for inclusion in the link pack area and the extended link pack area (continued)

Fortran module name	Description	RMODE
AFH5SPBP AFBDSPAP IFYDSPAP	VSF 1.4 array span calculator	ANY
AFH5TIMP AFBVTIMP	VSF obtain date and time routine	ANY
AFH5TRCH AFBVTRCH	VSF traceback routine	ANY
AFH5UNIN AFBVUNIN	VSF vector unnorm argument exception handler	ANY
AFH5VDMQ AFBVDUMQ IFYVDUMQ	VSF PDUMP/CPDUMP service routine	ANY
AFH5VINI AFBVVINI	VSF vector common area initializer	ANY
AFH5VIOS AFBVVIOS	VSF non-keyed VSAM routine	24
AFH5VTEN AFBVVTEN	VSF floating point conversion constants	ANY

Note: AFH5RENA, AFH5RENB, and AFH5RENP are used only for applications that were link-edited with VS FORTRAN Version 1 or 2 for execution in load mode.

Language Environment PL/I component modules

The PL/I component modules and *aliases* listed in Table 32 can be moved into LPA/ELPA using the sample job IBMALLP2 or IBMPLPA1 found in the SCEESAMP data set. The Language Environment base modules listed in Table 28 on page 247 should also be moved into LPA/ELPA.

Table 32. PL/I modules eligible for inclusion in the link pack area and the extended link pack area

PL/I module name	Description	RMODE
IBMREV10 CEEEV010	PL/I event handler	ANY
CEEEV011	Enterprise PL/I for z/OS event handler	ANY
IBMRCCLA IBMBCCLA	Conversion director (complex strings)	24
IBMRCRA IBMBCRA	Conversion director (non-complex strings)	24
IBMRCOMP	Conversion routines vector	24
IBMRDMPJ	Dump formatter for Japanese	ANY
IBMRDMPM	Dump formatter for mixed-case US English	ANY
IBMRDMPU	Dump formatter for uppercase English	ANY
IBMREDOA IBMEDO	Open diagnostic file module	24

Table 32. PL/I modules eligible for inclusion in the link pack area and the extended link pack area (continued)

PL/I module name	Description	RMODE
IBMREDTA IBMBEDTA	Diagnostic file transmitter	24
IBMREDWA IBMBEDWA	Console transmitter	24
IBMREMT IIBMMSGT	Message table	ANY
IBMREOCA IBMBOECA	ON-code module / ON-code calculator	ANY
IBMRKDBA IBMBKDBA	Dump file transmitter	24
IBMRKDOA IBMBKDOA	Open dump file	24
IBMRKDTA IBMBKDTA	Dump file transmitter	24
CEEKMRA IBMBKMRA IBMRKMRA	Link to main dump control module	24
IBMRKPTA IBMBKPTA	Dump parameter translate module	24
IBMLANA IBMBLANA	Language table (mixed-case US English)	24
IBMLANN IBMBLANN	Language table (Japanese)	24
IBMLANU IBMBLANU	Language table (uppercase English)	24
IBMLIB1	Lib pack (below the line)	24
IBMLNNTA IBMBLNTA	Language table (mixed-case US English)	24
IBMLNNTN IBMBLNTN	Language table (Japanese)	24
IBMLNNTU IBMBLNTU	Language table (uppercase English)	24
IBMRMCTA IBMBMCTA	ERF/ERFC (extended float)	24
IBMROCAA IBMBOCAA	Close module	24
IBMROPEA IBMBOPEA	Open routine (VSAM)	24
IBMROPZA IBMBOPZA	Direct output file formatter	24
IBMRPDBA IBMBPDBA	Debugger interface module	24
IBMRPESA IBMBPESA	ABEND analyzer	24

Table 32. PL/I modules eligible for inclusion in the link pack area and the extended link pack area (continued)

PL/I module name	Description	RMODE
IBMRPEVA IBMBPEVA	ABEND diagnostic message module	24
IBMRPTLA IBMBPTLA	Transient library level data	24
IBMRAAA IBMBRAAA	IBMRAAI: regional sequential output	24
IBMRRABA IBMBRABA	REG(1) sequential unbuffered transmitter	24
IBMRRACA IBMBRACA	BSAM LOAD REG(2) buffered F-format transmitter	24
IBMRRADA IBMBRADA	REG(2) SEQ. unbuffered transmitter	24
IBMRAEA IBMBRAEA	REG(3) buffered F-format transmitter	24
IBMRAFA IBMBRAFA	REG(3) sequential unbuffered F-format transmitter	24
IBMRRAGA IBMBRAGA	REG(3) buffered U+V-format transmitter	24
IBMRAHA IBMBRAHA	REG(3) sequential unbuffered U+V-format transmitter	24
IBMRAIA IBMBRAIA	REG(3) buffered VS-format transmitter	24
IBMRRBAA IBMBRBAA	BSAM REG(1) buffered F-format transmitter	24
IBMRRBBA IBMBRBBA	BSAM REG(1) unbuffered F-format transmitter	24
IBMRRBCA IBMBRBCA	REG(2)+(3) buffered F-format transmitter	24
IBMRRBDA IBMBRBDA	REG(2)+(3) unbuffered F-format transmitter	24
IBMRRBEA IBMBRBEA	BSAM REG(3) buffered U+V-format transmitter	24
IBMRRBFA IBMBRBFA	BSAM REG(3) update U+V-format transmitter	24
IBMRRBGA IBMBRBGA	BSAM REG(3) input/update VS-format transmitter	24
IBMRRCAA IBMBRCAA	BSAM (consecutive) F-format transmitter	24
IBMRCBA IBMRCBA	BSAM (consecutive) U-format transmitter	24
IBMRRCCA IBMBRCCA	BSAM (consecutive) V-format transmitter	24
IBMRRCDA IBMBRCDA	Consecutive unbuffered OMR transmitter	24

Table 32. PL/I modules eligible for inclusion in the link pack area and the extended link pack area (continued)

PL/I module name	Description	RMODE
IBMRRCEA IBMBRCEA	Consecutive unbuffered device associated F-format transmitter	24
IBMRRDAA IBMBRDAA	REG(1) direct F-format transmitter	24
IBMRRDBA IBMBRDBA	REG(2)+(3) direct F-format transmitter	24
IBMRRDCA IBMBRDCA	REG(3) direct U-format transmitter	24
IBMRRDDA IBMBRDDA	REG(3) direct V+VS-format transmitter	24
IBMRRCAA IBMBRCAA	Consecutive buffered record I/O error modules	24
IBMRRDAA IBMBRDAA	QISAM+BISAM record I/O error modules	24
IBMRRDCA IBMBRDCA	REG+SEQ+T.P. files record I/O error modules	24
IBMRRDAA IBMBRDAA	VSAM record I/O error modules	24
IBMRRDFA IBMBRDFA	Record I/O endfile module	24
IBMRRJAA IBMBRJAA	QISAM (SCAN) F-format transmitter	24
IBMRRJBA IBMBRJBA	QISAM (SCAN) V-format transmitter	24
IBMRRKAA IBMBRKAA	IBMRRKC: indexed direct non-exclusive	24
IBMRRKBA IBMBRKBA	BISAM F-format transmitter	24
IBMRRKCA IBMBRKCA	BISAM V-format transmitter	24
IBMRRLAA IBMBRLAA	QISAM (LOAD) F-format transmitter	24
IBMRRLBA IBMBRLBA	QISAM (LOAD) V-format transmitter	24
IBMRRQAA IBMBRQAA	QSAM F-format transmitter	24
IBMRRQBA IBMBRQBA	QSAM V-format transmitter	24
IBMRRQCA IBMBRQCA	QSAM U-format transmitter	24
IBMRRQDA IBMBRQDA	QSAM paper tape transmitter	24
IBMRRQEA IBMBRQEA	Buffered consecutive spanned record format input	24

Table 32. PL/I modules eligible for inclusion in the link pack area and the extended link pack area (continued)

PL/I module name	Description	RMODE
IBMRRQFA IBMBRQFA	Buffered consecutive spanned record format output	24
IBMRRQGA IBMBRQGA	Buffered consecutive record format update	24
IBMRRQHA IBMBRQHA	Consecutive buffered OMR	24
IBMRRQIA IBMBRQIA	Consecutive buffered device associated	24
IBMRRTPA IBMBRTPA	Teleprocessing buffered input/output files	24
IBMRRVAA IBMBRVAA	ESDS transmitter	24
IBMRRVGA IBMBRVGA	KSDS sequential output	24
IBMRRVHA IBMBRVHA	KSDS or PATH input/update/direct	24
IBMRRVIA IBMBRVIA	VSAM RRDS	24
IBMRRXAA IBMBRXAA	REG(1) direct F-format exclusive transmitter	24
IBMRRXBA IBMBRXBA	REG(2)+(3) direct F-format exclusive transmitter	24
IBMRRXCA IBMBRXCA	REG(3) direct U-format exclusive transmitter	24
IBMRRXDA IBMBRXDA	REG(3) direct V+VS-format exclusive transmitter	24
IBMRRYAA IBMBRYAA	BISAM F-format transmitter	24
IBMRRYBA IBMBRYBA	BISAM FB-format transmitter	24
IBMRRYCA IBMBRYCA	BISAM V-format transmitter	24
IBMRRYDA IBMBRYDA	BISAM VB-format transmitter	24
IBMRSAP IBMESAP	CICS bootstrap	24
IBMRSICA IBMBSICA	Conversational input transmitter	24
IBMRSOCA IBMBSOCA	Conversational output transmitter	24
IBMRSOFA IBMBSOFA	Output file transmitter (F-format)	24
IBMRSOUA IBMBSOUA	Output file transmitter (U-format)	24

Table 32. PL/I modules eligible for inclusion in the link pack area and the extended link pack area (continued)

PL/I module name	Description	RMODE
IBMRSOVA IBMBSOVA	Output file transmitter (V-format)	24
IBMRSPCA IBMBSPCA	Conversational file formatting	24
IBMRSTFA IBMBSTFA	Print file transmitter (F-record)	24
IBMRSTIA IBMBSTIA	Input file transmitter	24
IBMRSTUA IBMSTUA	Print file transmitter (U-record)	24
IBMRSTVA IBMSTVA	Print file transmitter (V-record)	24
IBMSOPAA IBMOPAA	Open	24
IBMUPJR0 IBMTPJRA	OS PL/I multitasking load module compatibility	24
IBM9LMSA	NLS mixed-case message source	ANY
IBM9LMSN	NLS Japanese message source	ANY
IBM9LMSU	NLS uppercase message source	ANY
IBM9LM2A	NLS mixed-case message	ANY
IBM9LM2N	NLS Japanese message	ANY
IBM9LM2U	NLS uppercase English message	ANY

Appendix D. National language support

This topic contains information to help you modify your code for national language support and list the codes for each country.

Modifying the JCL for Japanese national language support

Table 33 specifies additional changes you will need to make in the sample customization jobs if you want to install Language Environment Japanese national language support (NLS) on the MVS platform.

Table 33. JCL modifications for Japanese national language support

For this MVS job...	Modify the JCL like this...
CEEWDXIT CEEWCXIT CEEWUXIT	Change the NATLANG runtime option default in the CEEXOPT macro to NATLANG=(JPN).
IGZWMLP4	To store the Japanese module in the link pack area, remove the IGZCMGEN module name and add the IGZCMGJA module name.

National language support country codes for Language Environment

Table 34 contains valid country identifiers along with their respective countries:

Table 34. Country codes

Code	Country/region	Code	Country/region
AD	Andorra	AE	United Arab Emirates
AF	Afghanistan	AG	Antigua and Barbuda
AL	Albania	AN	Netherlands Antilles
AO	Angola	AR	Argentina
AT	Austria	AU	Australia
BA	Bosnia/ Herzegovina	BB	Barbados
BD	Bangladesh	BE	Belgium
BF	Burkina Faso (Upper Volta)	BG	Bulgaria
BH	Bahrain	BI	Burundi
BJ	Benin	BM	Bermuda
BN	Brunei Darussalam	BO	Bolivia
BR	Brazil	BS	Bahamas
BU	Burma	BW	Botswana
CA	Canada	CF	Central African Republic
CG	Congo	CH	Switzerland
CI	Ivory Coast	CL	Chile
CM	Cameroon	CN	People's Republic of China
CO	Colombia	CR	Costa Rica
CS	Czechoslovakia	CU	Cuba
CY	Cyprus	CZ	Czech Republic
DE	Germany	DK	Denmark
DO	Dominican Republic	DZ	Algeria
EC	Ecuador	EE	Estonia
EG	Egypt	ES	Spain
ET	Ethiopia	FI	Finland
FR	France	GA	Gabon

Table 34. Country codes (continued)

Code	Country/region	Code	Country/region
GB	United Kingdom	GH	Ghana
GM	Gambia	GN	Guinea
GR	Greece	GT	Guatemala
GW	Guinea-Bissau	GY	Guyana
HK	China (Hong Kong S.A.R.)	HN	Honduras
HR	Croatia	HT	Haiti
HU	Hungary	ID	Indonesia
IE	Ireland	IL	Israel
IN	India	IQ	Iraq
IR	Iran	IS	Iceland
IT	Italy	JM	Jamaica
JO	Jordan	JP	Japan
KE	Kenya	KR	Korea, Republic of
KW	Kuwait	KY	Cayman Islands
LB	Lebanon	LC	Saint Lucia
LI	Lichtenstein	LK	Sri Lanka
LR	Liberia	LS	Lesotho
LT	Lithuania	LU	Luxembourg
LV	Latvia	LY	Libya
MA	Morocco	MC	Monaco
MG	Madagascar	MK	Macedonia
ML	Mali	MO	China (Macau S.A.R.)
MR	Mauritania	MT	Malta
MU	Mauritius	MW	Malawi
MX	Mexico	MY	Malaysia
MZ	Mozambique	NA	Namibia
NC	New Caledonia	NG	Nigeria
NE	Niger	NI	Nicaragua
NL	Netherlands	NO	Norway
NZ	New Zealand	OM	Oman
PA	Panama	PE	Peru
PG	Papua New Guinea	PH	Philippines
PK	Pakistan	PL	Poland
PR	Puerto Rico	PT	Portugal
PY	Paraguay	QA	Qatar
RO	Romania	RU	Russian Federation
SA	Saudi Arabia	SC	Seychelles
SD	Sudan	SE	Sweden
SG	Singapore	SI	Slovenia
SK	Slovakia	SL	Sierra Leone
SN	Senegal	SO	Somalia
SR	Surinam	SU	Union of Soviet Socialist Republics
SV	El Salvador	SY	Syria
SZ	Swaziland	TD	Chad
TG	Togo	TH	Thailand
TN	Tunisia	TR	Turkey
TT	Trinidad and Tobago	TW	Taiwan
TZ	Tanzania	UG	Uganda
US	United States	UY	Uruguay
VE	Venezuela	VU	Vanuatu
WS	Western Samoa	YE	Yemen
YU	Yugoslavia	ZA	South Africa
ZM	Zambia	ZR	Zaire

Table 34. Country codes (continued)

Code	Country/region	Code	Country/region
ZW	Zimbabwe		

Appendix E. Accessibility

Accessible publications for this product are offered through the z/OS Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

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

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. 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.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually

exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!

(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Note:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
 2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
 3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.
- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

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

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

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

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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. 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.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (<http://www.ibm.com/software/support/systemsz/lifecycle/>)
 - For information about currently-supported IBM hardware, contact your IBM representative.
-

Programming interface information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Language Environment in z/OS. This publication also documents information that is NOT intended to be used as a Programming Interface of Language Environment.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml (<http://www.ibm.com/legal/copytrade.shtml>).

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

A

abend
 CEEEXTAN CSECT 152
 CICS 154
 customization job 153
 identifying 151
 non-CICS (MVS batch) 153
 syntax 195
 using 195
abnormal termination exit
 CEEEXTAN CSECT 152
 CEEWQEXT 154
 CICS 154
 customization job 153
 identifying 151
 non-CICS (MVS batch) 153
 syntax 195
 using 195
abnormal termination user exit 147
ABPERC runtime option 46
ABTERMENC runtime option 47
accessibility 277
 contact IBM 277
 features 277
AIXBLD runtime option 49
ALL31 runtime option 50
ANYHEAP runtime option 52
assembler language
 user exit 185
assembler user exit 147
assistive technologies 277
AUTOTASK runtime option 54

B

BELOWHEAP runtime option 54

C

cataloged procedure
 list of 159
 making available to your jobs 160
CBLOPTS runtime option 56
CBLPSHPOP runtime option 57
CBLQDA runtime option 58
CEEBDATX abnormal termination
 exit 151
CEEBXITA assembler user exit 185
CEEBXITA assembler user exit
 interface 188
CEECOPT 24
CEEDOPT 24
CEEPRMxx 23
CEEROPT 43
CEEWCEXT
 JCL for 154
CEEWCXIT 43
CEEWDEXT 43, 151
 JCL for 154
CEEWDXIT 43
CEEWHLLX 43, 150
 JCL for 151
CEEWLNUE
 JCL for 154
CEEWQEXT
 JCL for 154
CEEWROPT 43
CEEWUXIT 43
 JCL for 150
CEEXOPT macro 39
CELQDOPT 24
CHECK runtime option 61
CICS
 customizing
 abnormal termination exit 154
 installing Language Environment
 support for 163
COBOL
 compatibility with Language
 Environment options 45
 runtime options specific to 45
COBOL component modules 251
COBOL debug file name 178
COBOL debug file user exit 179
COBOL debug file user exit
 interface 179
COBOL formatted dump behavior 177
COBOL parameter list exit 175
COBOL reusable environment
 behavior 176
COBOL runtime environment 176
COBOL runtime environment
 behavior 178
command
 syntax diagrams xi
condition nesting 63
COUNTRY runtime option 61, 273
CSECT
 CEECOPT, CICS macro
 sample of 41
 CEEDOPT, non-CICS macro
 sample of 40, 42
customizing
 high-level language user exit 150
 procedure 3, 150, 151, 154, 155, 156,
 176, 181

D

D CEE 23
DCT (destination control table). 165
DEBUG runtime option 63
DEPTHCONDLMT runtime option 63
destination control table (DCT) 165
DYNDUMP runtime option 65

E

EDCLLOCL
 modifying the JCL 181

ELPA (extended link pack area) 249
ENVAR runtime option 68
ERRCOUNT runtime option 70
ERRUNIT runtime option 71
exit
 abnormal termination 151
 syntax 195
 assembler, customizing 185
 high-level language user 150

F

FILEHIST runtime option 72
FILETAG runtime option 72
Fortran
 customizing for Fortran
 applications 219, 240
LIBPACKS
 tailoring Fortran LIBPACKS 12,
 16, 240, 248

H

HEAP runtime option 74
HEAP64 runtime option 77
HEAPOOLS64 runtime option 84
high-level language (HLL) user exit 147
high-level language user exit 150, 185

I

IGZWARRE
 modifying the JCL 178
IMS
 performance considerations 171
INQPCOPN runtime option 88
installation
 support for CICS 163
INTERRUPT runtime option 88
IOHEAP64 runtime option 89

J

JCL (Job Control Language)
 common modifications
 for customization 3
JCL for CEEWCEXT 154
JCL for CEEWDEXT 154
JCL for CEEWHLLX 151
JCL for CEEWLNUE 156
JCL for CEEWQEXT 154
JCL for CEEWUXIT 150
JCL for EDCLLOCL 181
JCL for IGZWAPSX 176
JCL for IGZWARRE 178
Job Control Language (JCL)
 common modifications
 for customization 3

K

- keyboard
 - navigation 277
 - PF keys 277
 - shortcut keys 277

L

- Language Environment
 - customizing 3
- library 159
- library routine retention 171
- LIBSTACK runtime option 92
- link pack area (LPA).
 - installing Language Environment into 249
- load notification user exit 147
- locale time information 181
- LPA (link pack area)
 - installing Language Environment into 249

M

- modifying the 175
- MSGFILE runtime option 94
- MSGQ runtime option 98
- Multiple Virtual System (MVS)
 - installing
 - in a link pack area 249

N

- national language support (NLS) 273
- NATLANG runtime option 98
- navigation
 - keyboard 277
- nested conditions
 - limiting 63
- nested enclave behavior 177
- Notices 281

O

- OCSTATUS runtime option 100
- OS/VS COBOL 174
- OS/VS COBOL compatibility library
 - routines 173

P

- parmlib member
 - CEEPRMxx 23
 - examples 26
 - setting defaults 23
- passing
 - parameters at invocation 45, 56
 - runtime options at invocation 45, 56
- PC runtime option 101
- performance
 - considerations for IMS 171
- procedure, cataloged
 - list of 159
 - making available to your jobs 160

- program resource definitions
 - adding for CICS 163
- PRTUNIT runtime option 105
- PUNUNIT runtime option 105

R

- RDRUNIT runtime option 106
- RECPAD runtime option 106
- RTEREUS runtime option 110
- runtime options 45
 - ABPERC—percolate an abend 46
 - ABTERMENC—determine how an enclave terminates 47
 - AIXBLD—invoke AMS for COBOL 49
 - ALL31—indicate whether application runs in AMODE(31) 50
 - ANYHEAP—control unrestricted library heap storage 52
 - AUTOTASK—specify whether Fortran MTF is to be used 54
 - BELOWHEAP—control library heap storage below 16 MB 54
 - CBLOPTS—specify format of COBOL argument 56
 - CBLPSPHPOP—control CICS commands 57
 - CBLQDA—control COBOL QSAM 58
 - CHECK—detect checking errors 61
 - COBOL-specific 45
 - COUNTRY—specify default date/time formats 61
 - customizing 19
 - DEBUG—activate COBOL batch debugging 63
 - DEPTHCONDLMT—limit extent of nested conditions 63
 - ENVAR—set initial values for environment variables 68
 - ERRCOUNT—specify number of errors allowed 70
 - ERRUNIT—specify unit number to which error information is directed 71
 - FILEHIST—specify whether to allow a file definition to be changed at runtime 72
 - FILETAG—specify whether to allow AUTOTAG / AUTOCVT. 72
 - HEAP—control allocation of heaps 74
 - HEAP64 — controls allocation of user heap storage 77
 - HEAPCHK—runs additional heap check tests 79
 - HEAPPOOLS—control allocation of optional heap pools storage 81
 - HEAPPOOLS64 — controls optional user heap storage management algorithm 84
 - INFOMSGFILTER—eliminates unwanted informational messages 86

runtime options (*continued*)

- INQPCOPN—control value in OPENED specifier of INQUIRE by unit statement 88
- INTERRUPT—cause attentions to be recognized by Language Environment 88
- IOHEAP64 — controls allocation of I/O heap storage 89
- LIBSTACK—control library stack storage 92
- MSGFILE—specify ddname of diagnostic file 94
- MSGQ—specify number of ISI blocks allocated 98
- NATLANG—specify national language 98
- OCSTATUS—control checking of file existence and whether file deletion occurs 100
- PC—control whether Fortran status common blocks are shared among load modules 101
- PLITASKCOUNT—control the maximum number of active tasks 102
- POSIX—specify whether enclave runs with POSIX semantics 103
- PROFILE—controls optional PROFILE use 104
- PRTUNIT—specifies unit number used for PRINT and WRITE statements 105
- PUNUNIT—specifies unit number used for PUNCH statements 105
- RDRUNIT—specifies unit number used for READ statements 106
- RECPAD—specifies whether a formatted input record is padded with blanks 106
- RTEREUS—initialize a reusable COBOL environment 110
- SIMVRD—specify VSAM KSDS for COBOL 112
- STACK—allocate stack storage 113
- STORAGE—control storage 118
- TERMTHDACT—specify type of information generated with unhandled error 121
- TEST—indicate debug tool to gain control 127
- THREADHEAP—control the allocation of thread-level heap storage 129
- THREADSTACK—control the allocation of stack storage 131
- TRACE—activate Language Environment runtime library tracing 135
- TRAP—handle abends and program interrupts 137
- UPSI—set UPSI switches. 140
- USRHDLR—register a user condition handler at stack frame 0 141
- VCTRSVAVE—use vector facility 143
- XUFLOW—specify program interrupt due to exponent underflow 143

S

sample job
 high-level language user exit 150
 procedure 3, 150, 151, 154, 155, 156,
 176, 181
sending comments to IBM xv
SET CEE 23
SETCEE 23
shortcut keys 277
SIMVRD runtime option 112
STACK runtime option 113
storage
 required for MVS
 link pack area for MVS 249
STORAGE runtime option 118
storage tuning user exit 147
 creating 158
summary of changes
 as updated December 2013 xvii
Summary of changes xvii
syntax diagrams
 how to read xi

T

target library
 description of 5
TERMTHDACT runtime option 121
TEST runtime option 127
THREADSTACK runtime option 131
TRACE runtime option 135
trademarks 283
TRAP runtime option 137
TSO/E LOGON procedure 159

U

UPSI runtime option 45
 ABPERC—percolate an abend 46
 ABTERMENC—determine how an
 enclave terminates 47
 AIXBLD—invoke AMS for
 COBOL 49
 ALL31—indicate whether application
 runs in AMODE(31) 50
 ANYHEAP—control unrestricted
 library heap storage 52
 AUTOTASK—specify whether Fortran
 MTF is to be used 54
 BELOWHEAP—control library heap
 storage below 16 MB 54
 CBLOPTS—specify format of COBOL
 argument 56
 CBLPSHPOP—control CICS
 commands 57
 CBLQDA—control COBOL
 QSAM 58
 CHECK—detect checking errors 61
 COBOL-specific 45
 COUNTRY—specify default date/time
 formats 61
 DEBUG—activate COBOL batch
 debugging 63
 DEPTHCONDLMT—limit extent of
 nested conditions 63

UPSI runtime option (*continued*)
 ENVAR—set initial values for
 environment variables 68
 ERRCOUNT—specify number of
 errors allowed 70
 ERRUNIT—specify unit number to
 which error information is
 directed 71
 FILEHIST—specify whether to allow a
 file definition to be changed at
 runtime 72
 FILETAG—specify whether to allow
 AUTOTAG / AUTOCVT. 72
 HEAP—control allocation of
 heaps 74
 HEAPCHK—runs additional heap
 check tests 79
 HEAPPOOLS—control allocation of
 optional heap pools storage 81
 INFMSGFILTER—eliminates
 unwanted informational
 messages 86
 INQPCOPN—control value in
 OPENED specifier of INQUIRE by
 unit statement 88
 INTERRUPT—cause attentions to be
 recognized by Language
 Environment 88
 LIBSTACK—control library stack
 storage 92
 MSGFILE—specify ddname of
 diagnostic file 94
 MSGQ—specify number of ISI blocks
 allocated 98
 NATLANG—specify national
 language 98
 OCSTATUS—control checking of file
 existence and whether file deletion
 occurs 100
 PC—control whether Fortran status
 common blocks are shared among
 load modules 101
 PLITASKCOUNT—control the
 maximum number of active
 tasks 102
 POSIX—specify whether enclave runs
 with POSIX semantics 103
 PROFILE—controls optional PROFILE
 use 104
 PRTUNIT—specifies unit number
 used for PRINT and WRITE
 statements 105
 PUNUNIT—specifies unit number
 used for PUNCH statements 105
 RDRUNIT—specifies unit number
 used for READ statements 106
 RECPAD—specifies whether a
 formatted input record is padded
 with blanks 106
 RTEREUS—initialize a reusable
 COBOL environment 110
 SIMVRD—specify VSAM KSDS for
 COBOL 112
 STACK—allocate stack storage 113
 STORAGE—control storage 118

UPSI runtime option (*continued*)
 TERMTHDACT—specify type of
 information generated with
 unhandled error 121
 TEST—indicate debug tool to gain
 control 127
 THREADHEAP—control the
 allocation of thread-level heap
 storage 129
 THREADSTACK—control the
 allocation of stack storage 131
 TRACE—activate Language
 Environment runtime library
 tracing 135
 TRAP—handle abends and program
 interrupts 137
 UPSI—set UPSI switches. 140
 USRHDLR—register a user condition
 handler at stack frame 0 141
 VCTRSVAVE—use vector facility 143
 XUFLOW—specify program interrupt
 due to exponent underflow 143
user
 exit
 abnormal termination 147
 assembler 147, 185
 high-level language 150, 185
 high-level language (HLL) 147
 load notification 147
 storage tuning 147
user interface
 ISPF 277
 TSO/E 277
USRHDLR runtime option 141

V

VCTRSVAVE runtime option 143

W

worksheet
 changing runtime option defaults
 on MVS 19

X

XUFLOW runtime option 143



Product Number: 5650-ZOS

Printed in USA

SA38-0685-01

