z/OS

Language Environment Programming Reference

Version 2 Release 1

SA38-0683-00

Note

Before using this information and the product it supports, read the information in “Notices” on page 495.

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.

Contents

Figures

Tables

About this document

This document supports z/OS (5650-ZOS).

IBM® z/OS Language Environment (also called Language Environment) provides common services and language-specific routines in a single run-time environment for C, C++, COBOL, Fortran (z/OS only; no support for z/OS UNIX System Services or CICS®), PL/I, and assembler applications. It offers consistent and predictable results for language applications, independent of the language in which they are written.

Language Environment is the prerequisite run-time environment for applications generated with the following IBM compiler products:
- z/OS XL C/C++ (feature of z/OS)
- z/OS® C/C++
- OS/390® C/C++
- C/C++ for MVS/ESA
- C/C++ for z/VM®
- XL C/C++ for z/VM
- AD/Cycle C/370™
- VisualAge for Java, Enterprise Edition for OS/390
- Enterprise COBOL for z/OS
- Enterprise COBOL for z/OS and OS/390
- COBOL for OS/390 & VM
- COBOL for MVS & VM (formerly COBOL/370)
- Enterprise PL/I for z/OS
- Enterprise PL/I for z/OS and OS/390
- VisualAge® PL/I
- PL/I for MVS & VM (formerly PL/I MVS™ & VM)
- VS FORTRAN and FORTRAN IV (in compatibility mode)

Although not all compilers listed are currently supported, Language Environment® supports the compiled objects that they created.

Language Environment supports, but is not required for, an interactive debug tool for debugging applications in your native z/OS environment.

Debug Tool is also available as a standalone product. Debug Tool Utilities and Advanced Functions is also available. For more information about Debug Tool, see http://www.ibm.com/software/awdtools/debugtool/.

Language Environment supports, but is not required for, VS FORTRAN Version 2 compiled code (z/OS only).

Language Environment consists of the common execution library (CEL) and the runtime libraries for C/C++, COBOL, Fortran, and PL/I.

For more information about VisualAge for Java, Enterprise Edition for OS/390, program number 5655-JAV, see the product documentation.

This publication provides application programmers with a detailed description of each Language Environment runtime option and callable service, as well as information about how to use them. It also provides programming examples that illustrate how each callable service can be used in routines written in Language Environment-conforming high-level languages (HLLs) and assembler language. Before using Language Environment, you should be familiar with the HLLs in which your applications are written. You should also understand the operating systems and any subsystems in which you plan to run Language Environment applications.

Using your documentation

The publications provided with Language Environment are designed to help you:
- Manage the runtime environment for applications generated with a Language Environment-conforming compiler.
- Write applications that use the Language Environment callable services.
- Develop interlanguage communication applications.
- Customize Language Environment.
- Debug problems in applications that run with Language Environment.
- Migrate your high-level language applications to Language Environment.

Language programming information is provided in the supported high-level language programming manuals, which provide language definition, library function syntax and semantics, and programming guidance information.

Each publication helps you perform different tasks, some of which are listed in Table 1.

Table 1. How to use z/OS Language Environment publications

To ...	Use ...
Evaluate Language Environment	*z/OS Language Environment Concepts Guide*
Plan for Language Environment	*z/OS Language Environment Concepts Guide* *z/OS Language Environment Runtime Application Migration Guide*
Install Language Environment	*z/OS Program Directory*
Customize Language Environment	*z/OS Language Environment Customization*
Understand Language Environment program models and concepts	*z/OS Language Environment Concepts Guide* *z/OS Language Environment Programming Guide* *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*
Find syntax for Language Environment runtime options and callable services	*z/OS Language Environment Programming Reference*
Develop applications that run with Language Environment	*z/OS Language Environment Programming Guide* and your language programming guide
Debug applications that run with Language Environment, diagnose problems with Language Environment	*z/OS Language Environment Debugging Guide*
Get details on runtime messages	*z/OS Language Environment Runtime Messages*
Develop interlanguage communication (ILC) applications	*z/OS Language Environment Writing Interlanguage Communication Applications* and your language programming guide

Table 1. How to use z/OS Language Environment publications (continued)

To ...	Use ...
Migrate applications to Language Environment	*z/OS Language Environment Runtime Application Migration Guide* and the migration guide for each Language Environment-enabled language

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 2. Syntax examples

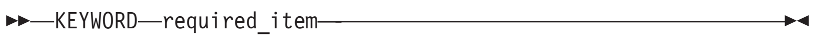
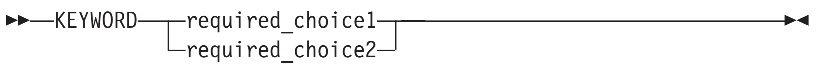
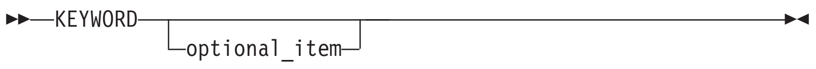
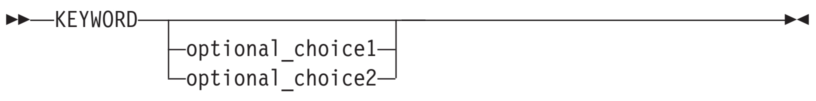
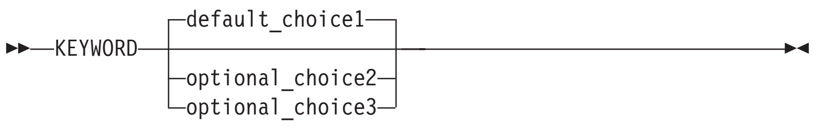
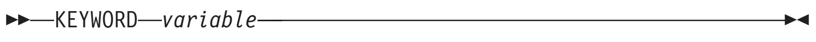
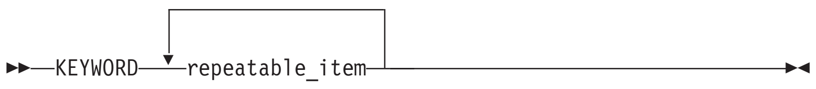
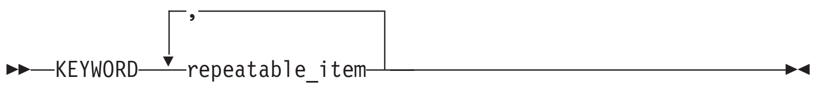
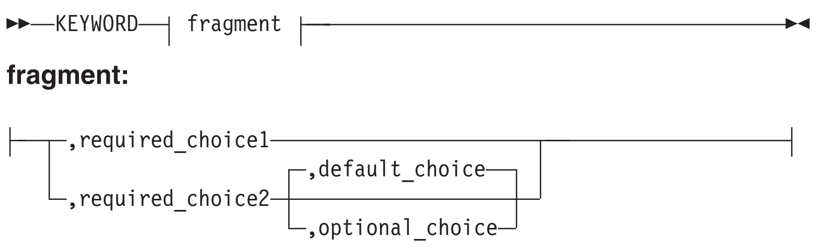
Item	Syntax example
Required item. Required items appear on the main path of the horizontal line. You must specify these items.	
Required choice. A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.	
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.	 default_choice1 KEYWORD optional_choice2 optional_choice3
Variable. Variables appear in lowercase italics. They represent names or values.	

Table 2. 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.	 KEYWORD fragment fragment: ,required_choice1 ,required_choice2 ,default_choice ,optional_choice

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, see 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 Programming Reference
  SA38-0683-00
- 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*

Part 1. Language Environment runtime options

The following sections contain detailed information about how to use the Language Environment runtime options.

Chapter 1. Summary of Language Environment runtime options

The quick reference tables for each of the Language Environment runtime options and information help you read syntax diagrams and specify the runtime options. The tables list the location of the options and briefly state their function.

Quick reference table for AMODE 31 runtime options

Table 3 provides a quick reference of the Language Environment runtime options for AMODE 31 applications.

Table 3. Quick reference table for runtime options - AMODE 31

Runtime options	Function
ABPERC	Percolates a specified abend. See "ABPERC" on page 9.
ABTERMENC	Sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater. See "ABTERMENC" on page 10.
AIXBLD ǀ NOAIXBLD	Invokes the access method services (AMS) for VSAM indexed and relative data sets to complete the file and index definition procedures for COBOL routines. See "AIXBLD (COBOL only)" on page 11.
ALL31	Indicates whether an application does or does not run entirely in AMODE(31). See "ALL31" on page 12.
ANYHEAP	Controls allocation of library heap storage not restricted to below the 16M line. See "ANYHEAP" on page 14.
ARGPARSE	Specifies if arguments on the command line are to be parsed in the usual C format. See "ARGPARSE ǀ NOARGPARSE (C only)" on page 15.
AUTOTASK	Specifies if Fortran Multitasking Facility is to be used by your program and the number of tasks that are allowed to be active. See "AUTOTASK ǀ NOAUTOTASK (Fortran only)" on page 16.
BELOWHEAP	Controls allocation of library heap storage below the 16M line. See "BELOWHEAP" on page 17.
CBLOPTS	Specifies the format of the argument string on application invocation when the main program is COBOL. See "CBLOPTS (COBOL only)" on page 18.
CBLPSHPOP	Controls if CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL subprogram is called. See "CBLPSHPOP (COBOL only)" on page 18.
CBLQDA	Controls COBOL QSAM dynamic allocation. See "CBLQDA (COBOL only)" on page 19.
CEEDUMP	Specifies options to control the processing of the Language Environment dump report. See "CEEDUMP" on page 19.
CHECK	Indicates if checking errors within an application should be detected. See "CHECK (COBOL only)" on page 22.
COUNTRY	Specifies the default formats for date, time, currency symbol, decimal separator, and the thousands separator based on a country. See "COUNTRY" on page 22.
DEBUG ǀ NODEBUG	Activates the COBOL batch debugging features specified by the "debugging lines" or the USE FOR DEBUGGING declarative. See "DEBUG (COBOL only)" on page 24.

Table 3. Quick reference table for runtime options - AMODE 31 (continued)

Runtime options	Function
DEPTHCONDLMT	Limits the extent to which conditions can be nested. See "DEPTHCONDLMT" on page 24.
DYNDUMP	Provides a way to obtain IPCS readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement. See "DYNDUMP" on page 26.
ENV	Specifies the operating system that your C application runs under. See "ENV (C only)" on page 28.
ENVAR	Sets the initial values for the environment variables. See "ENVAR" on page 29.
ERRCOUNT	Specifies how many conditions of severity 2, 3, and 4 can occur per thread before an enclave terminates abnormally. See "ERRCOUNT" on page 31.
ERRUNIT	Identifies the unit number to which runtime error information is to be directed. See "ERRUNIT (Fortran only)" on page 32.
EXECOPS	Specifies if runtime options can be specified on the command line. See "EXECOPS | NOEXECOPS (C only)" on page 32.
FILEHIST	FILEHIST specifies whether to allow the file definition of a file referred to by a ddname to be changed during run time. See "FILEHIST (Fortran only)" on page 33.
FILETAG	FILETAG ensures control of untagged z/OS UNIX files and standard streams terminal files when set up for conversion, and for control of certain open functions which tag new or empty z/OS UNIX files. See "FILETAG (C/C++ only)" on page 33.
FLOW | NOFLOW	Controls the FLOW output produced by OS/VS COBOL programs. See "FLOW (COBOL only)" on page 35.
HEAP	Controls allocation of the user heap. See "HEAP" on page 36.
HEAPCHK	Specifies that user heap storage can be inspected for damage and request a heap storage diagnostics report. See "HEAPCHK" on page 40.
HEAPPOOLS	Specifies that (Quick) heap storage manager can be used. See "HEAPPOOLS (C/C++ and Enterprise PL/I only)" on page 42.
HEAPZONES	Turns on overlay toleration and checking for user heaps. See "HEAPZONES" on page 46.
INFOMSGFILTER	Allows the user to eliminate unwanted informational messages. See "INFOMSGFILTER" on page 47.
INQPCOPN | NOINQPCOPN	INQPCOPN controls if 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. See "INQPCOPN (Fortran only)" on page 48.
INTERRUPT	Causes attentions recognized by the host operating system to be recognized by Language Environment. See "INTERRUPT" on page 49.
LIBSTACK	Controls the allocation of the thread's library stack storage. See "LIBSTACK" on page 52.
MSGFILE	Specifies the *ddname* of the runtime diagnostics file. See "MSGFILE" on page 53.
MSGQ	Specifies the number of ISI blocks allocated on a per-thread basis during execution. See "MSGQ" on page 56.
NATLANG	Specifies the national language to use for the runtime environment. See "NATLANG" on page 57.
OCSTATUS | NOOCSTATUS	Controls if the OPEN and CLOSE status specifiers are verified. See "OCSTATUS (Fortran only)" on page 58.
PAGEFRAMESIZE	Indicates the preferred page frame size for certain types of storage requests. Note the statement of direction associated with this runtime option. See "PAGEFRAMESIZE" on page 59.

Table 3. Quick reference table for runtime options - AMODE 31 (continued)

Runtime options	**Function**
PC	Specifies that Fortran static common blocks are not shared among load modules. See “PC (Fortran only)” on page 63.
PLIST	Specifies the format of the invocation arguments received by your C application when it is invoked. See “PLIST (C only)” on page 63.
PLITASKCOUNT	Controls the maximum number of tasks active at one time while you are running PL/I MTF applications. See “PLITASKCOUNT (PL/I only)” on page 65.
POSIX	Specifies if the enclave can run with the POSIX semantics. See “POSIX” on page 65.
PROFILE	Controls the use of an optional profiler which collects performance data for the running application. See “PROFILE” on page 67
PRTUNIT	Identifies the unit number used for PRINT and WRITE statements that do not specify a unit number. See “PRTUNIT (Fortran only)” on page 67.
PUNUNIT	Identifies the unit number used for PUNCH statements that do not specify a unit number. See “PUNUNIT (Fortran only)” on page 68.
RDRUNIT	Identifies the unit number used for READ statements that do not specify a unit number. See “RDRUNIT (Fortran only)” on page 68.
REDIR	Specifies if redirections for `stdin`, `stderr`, and `stdout` are allowed from the command line. See “REDIR ǀ NOREDIR (C only)” on page 69.
RECPAD	Specifies if a formatted input record is padded with blanks. See “RECPAD (Fortran only)” on page 68.
RPTOPTS	Specifies that a report of the runtime options in use by the application be generated. See “RPTOPTS” on page 69
RPTSTG	Specifies that a report of the storage used by the application be generated at the end of execution. See “RPTSTG” on page 71
RTEREUS ǀ NORTEREUS	Initializes the runtime environment to be reusable when the first COBOL program is invoked. See “RTEREUS (COBOL only)” on page 72.
SIMVRD ǀ NOSIMVRD	Specifies if your COBOL programs use a VSAM KSDS to simulate variable length relative organization data sets. See “SIMVRD (COBOL only)” on page 74.
STACK	Controls the allocation and management of stack storage. See “STACK” on page 75.
STORAGE	Controls the contents of storage that is allocated and freed. See “STORAGE” on page 79.
TERMTHDACT	Sets the level of information produced due to an unhandled error of severity 2 or greater. See “TERMTHDACT” on page 82.
TEST ǀ NOTEST	Specifies that a debug tool is to be given control according to the suboptions specified. See See “TEST ǀ NOTEST” on page 87.
THREADHEAP	Controls the allocation and management of thread-level heap storage. See “THREADHEAP” on page 89.
THREADSTACK	Controls the allocation of thread-level stack storage for both the upward and downward-growing stack. See “THREADSTACK” on page 91.
TRACE	Determines if Language Environment runtime library tracing is active. See “TRACE” on page 94.
TRAP	Specifies how Language Environment routines handle abends and program interrupts. See “TRAP” on page 96.
UPSI	Sets the eight UPSI switches on or off. See “UPSI (COBOL only)” on page 98.
USRHDLR	Registers user condition handlers. See “USRHDLR ǀ NOUSRHDLR” on page 99.
VCTRSAVE	Specifies if any language in an application uses the vector facility when user-written condition handlers are called. See “VCTRSAVE” on page 100.

Table 3. Quick reference table for runtime options - AMODE 31 (continued)

Runtime options	Function
XPLINK	Controls the initialization of the XPLINK environment. See "XPLINK" on page 101.
XUFLOW	Specifies if an exponent underflow causes a program interrupt. See "XUFLOW" on page 103.

Quick reference table for AMODE 64 runtime options

Table 4 provides a quick reference of the Language Environment runtime options for AMODE 64 applications.

Table 4. Quick reference table for runtime options - AMODE 64

Runtime options	Function
ARGPARSE	Specifies if arguments on the command line are to be parsed in the usual C format. See "ARGPARSE | NOARGPARSE (C only)" on page 15.
CEEDUMP	Specifies options to control the processing of the Language Environment dump report. See "CEEDUMP" on page 19.
DYNDUMP	Provides a way to obtain IPCS readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement. See "DYNDUMP" on page 26.
ENVAR	Sets the initial values for the environment variables. See "ENVAR" on page 29.
EXECOPS	Specifies whether runtime options can be specified on the command line. See "EXECOPS | NOEXECOPS (C only)" on page 32.
FILETAG	FILETAG ensures control of untagged z/OS UNIX files and standard streams terminal files when set up for conversion, and for control of certain open functions which tag new or empty z/OS UNIX files. See "FILETAG (C/C++ only)" on page 33.
HEAP64	Controls allocation of the user heap. See "FILETAG (C/C++ only)" on page 33. See "FILETAG (C/C++ only)" on page 33.
HEAPCHK	Specifies that user heap storage can be inspected for damage and request a heap storage diagnostics report. See "HEAPCHK" on page 40.
HEAPPOOLS	Specifies that the (Quick) heap storage manager can be used to manage user heap storage above the 16M line and below the 2G bar. See "HEAPPOOLS (C/C++ and Enterprise PL/I only)" on page 42.
HEAPPOOLS64	Specifies that the (Quick) heap storage manager can be used to manage user heap storage above the 2G bar. See "HEAPPOOLS64 (C/C++ and AMODE 64 only)" on page 44.
HEAPZONES	Turns on overlay toleration and checking for user heaps. See "HEAPZONES" on page 46.
INFOMSGFILTER	Allows the user to eliminate unwanted informational messages. See "INFOMSGFILTER" on page 47.
IOHEAP64	Controls allocation of I/O heap storage. See "IOHEAP64 (AMODE 64 only)" on page 49.
LIBHEAP64	Controls allocation of library heap storage. See "LIBHEAP64 (AMODE 64 only)" on page 51.
NATLANG	Specifies the national language to use for the runtime environment. See "NATLANG" on page 57.
PAGEFRAMESIZE64	Indicates the preferred frame size for certain types of storage requests. See "PAGEFRAMESIZE64" on page 61.
POSIX	Specifies if the enclave can run with the POSIX semantics. See "POSIX" on page 65.

Table 4. Quick reference table for runtime options - AMODE 64 (continued)

Runtime options	Function
PROFILE	Controls the use of an optional profiler which collects performance data for the running application. See "PROFILE" on page 67.
REDIR	Specifies if redirections for `stdin`, `stderr`, and `stdout` are allowed from the command line. See "REDIR | NOREDIR (C only)" on page 69.
RPTOPTS	Specifies that a report of the runtime options in use by the application be generated. See "RPTOPTS" on page 69.
RPTSTG	Specifies that a report of the storage used by the application be generated at the end of execution. See "RPTSTG" on page 71.
STACK64	Controls the allocation and management of stack storage. See "STACK64 (AMODE 64 only)" on page 78.
STORAGE	Controls the contents of storage that is allocated and freed. See "STORAGE" on page 79.
TERMTHDACT	Sets the level of information produced due to an unhandled error of severity 2 or greater. See "TERMTHDACT" on page 82.
TEST | NOTEST	Specifies that a debug tool is to be given control according to the suboptions specified. See "TEST | NOTEST" on page 87.
THREADSTACK64	Controls the allocation of thread-level stack storage for both the upward and downward-growing stack. See "THREADSTACK64 (AMODE 64 only)" on page 93.
TRACE	Determines if Language Environment runtime library tracing is active. See "TRACE" on page 94.
TRAP	Specifies how Language Environment routines handle abends and program interrupts. See "TRAP" on page 96.

How to specify runtime options

When specifying runtime options, use commas to separate any suboptions of those options. If you do not specify a suboption, you must still include the comma to indicate the omission of the suboption. For example: `STACK(,,ANYWHERE,FREE,,)`. Trailing commas, however, are not required. `STACK(,,ANYWHERE,FREE)` is valid. If you do not specify any suboptions, they are ignored. Either of the following is valid: `STACK` or `STACK()`.

If you run applications that are invoked by one of the `exec` family of functions, such as a program executed under the z/OS UNIX System Services shell, you can also use the _CEE_RUNOPTS environment variable to specify runtime options.

Refer to *z/OS Language Environment Programming Guide* for a detailed description of the various ways in which you can specify Language Environment runtime options and program arguments, and use _CEE_RUNOPTS.

Restriction: The double quotes character (") may not be part of the runtime options string for applications running under Turkish code page 1026. Use the single quote character (') instead.

Propagating runtime options with spawn and exec

When going from AMODE 31 to AMODE 64 and back through the `spawn` or `exec` functions, Language Environment rebuilds the _CEE_RUNOPTS environment variable as a way to propagate runtime options to the new program. In the situations where AMODE 31 specific options or AMODE 64 specific options would be passed across to the other mode, Language Environment ignores these options. No messages are issued. For example, when the STACK option is sent across from AMODE 31 to AMODE 64, it is ignored. This is because the AMODE 64 application uses the STACK64 option. No attempt to convert the AMODE 31 option to the new AMODE 64 option is performed.

Chapter 2. Using the Language Environment runtime options

The Language Environment runtime options, their syntax, and their usage are described. IBM-supplied default keywords appear **above** the main path or options path in the syntax diagrams. In the parameter list, IBM-supplied default choices are underlined. The minimum unambiguous abbreviation of each Language Environment option is also indicated in its syntax diagram with capital letters (for example, ABPerc indicates that ABP is the minimum abbreviation).

ABPERC

Derivation: ABnormal PERColation

ABPERC percolates an abend whose code you specify. This provides Language Environment condition handling semantics for all abends, except for the one specified. TRAP(ON) must be in effect. When you run with ABPERC and encounter the specified abend:

- User condition handlers are not enabled.
- Under 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 formatted 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 that are opened by HLLs are not closed by Language Environment, so records might be lost.
- Resources that are acquired by Language Environment are not freed.
- A debugging tool, if one is active, 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*.

The default value for non-CICS applications is ABPERC(NONE).

ABPERC is ignored under CICS.

Syntax

```
                ┌─NONE───┐
►►──ABPerc──(───┴─abcode─┴──)──────────────────────────────────►◄
```

NONE
: Specifies that all abends are handled according to Language Environment condition handling semantics.

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

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

U*dddd* A user abend code, where *dddd* is a decimal user abend code. Any 4-character string can also be used as *dddd*.

z/OS UNIX consideration

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

Usage notes

- Language Environment ignores ABPERC(S0C*x*). In this instance, no abend is percolated, and Language Environment condition handling semantics are in effect.
- You can identify only one abend code with this option. However, an abend U0000 is interpreted in the same way as S000.

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.

The default value for non-CICS applications is ABTERMENC(ABEND).

The default value for CICS applications is ABTERMENC(ABEND).

Syntax

```
                       ┌─ABEND───┐
►►──ABTermenc──(───────┴─RETCODE─┴──)──────────────────────────────────►◄
```

ABEND
When an unhandled condition of severity 2 or greater is encountered, Language Environment issues an abend to end the enclave. Default abend processing occurs as follows:

- Language Environment sets an abend code value that depends on the type of unhandled condition.
- Language Environment sets a reason code value that depends on the type of unhandled condition.
- Language Environment does not request a system dump.
- Language Environment issues an abend to terminate the task.

RETCODE
When an unhandled condition of severity 2 or greater is encountered, Language Environment issues a return code and reason code to end the enclave.

z/OS UNIX consideration

In a multithreaded application with ABTERMENC(ABEND), Language Environment issues an abend on the task associated with the initial processing thread (IPT), regardless of which thread experienced the unhandled condition. All non-IPT threads are terminated normally. This means that the thread that encountered the unhandled condition is terminated normally if it is a non-IPT thread.

Usage notes

- You can use the assembler user exit, CEEAUE_ABND, to modify the behavior of this runtime option by setting the CEEAUE_ABND flag. See *z/OS Language Environment Programming Guide* for more information.
- To gather information about the unhandled condition, see "TERMTHDACT" on page 82.

For more information

For information about return code calculation and abend codes, 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.

The default value for non-CICS applications is NOAIXBLD.

AIXBLD is ignored under CICS.

Syntax

```
   ┌─NOAIXBLD─┐
►►─┼─NOAIX────┼──────────────────────────────────────────────►◄
   ├─AIXBLD───┤
   └─AIX──────┘
```

NOAIXBLD|NOAIX
Does not invoke the access method services for VSAM indexed and relative data sets. NOAIXBLD can be abbreviated NOAIX.

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

z/OS UNIX consideration

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

Performance consideration

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.

For more information

- 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).
- See "MSGFILE" on page 53 for information about the MSGFILE runtime option.

ALL31

Derivation: ALL AMODE 31

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

Guideline: ALL31 should 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.

The default value for non-CICS applications is ALL31(ON).

The default value for CICS applications is ALL31(ON).

Syntax

```
              ┌─ON──┐
►►──ALL31──(──┼─────┼──)──────────────────────────────────────────►◄
              └─OFF─┘
```

ON If the initial routine of the Language Environment application is AMODE 31, this setting indicates that no other routines in the application will be AMODE 24.

If the initial routine of the Language Environment application is AMODE 24, ALL31 is dynamically switched to OFF. No message will be issued to indicate this action. However, if you generate a Language Environment runtime options report using the RPTOPTS runtime option, the ALL31 option will be reported as `"Override"` under the `LAST WHERE SET` column.

When ALL31(ON) remains in effect:

- 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.
- Language Environment allocates storage for the common anchor area (CAA) and other control blocks in unrestricted storage.
- COBOL EXTERNAL data is allocated in unrestricted storage.

`OFF`

Indicates that one or more routines of a Language Environment application are AMODE 24. When ALL31(OFF) is in effect:

- Language Environment uses more storage below the 16M line.
- 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.
- Language Environment allocates storage for the common anchor area (CAA) and other control blocks in storage below the 16M line.
- COBOL EXTERNAL data is allocated in storage below the 16M line.

Restriction: If you use the setting ALL31(OFF), you must also use the setting STACK(,,BELOW,,,). AMODE 24 routines require stack storage below the 16M line.

z/OS UNIX considerations

- In a multithreaded environment, the ALL31 option applies to all threads in a process.
- In a multithreaded environment, the thread start routine specified in the C pthread_create() function call is invoked in AMODE 31.

Usage notes

- **Restrictions:** You must specify ALL31(OFF) if your COBOL applications contain one of the following programs:
  - VS COBOL II NORES
  - OS/VS COBOL (non-CICS)
  - If the Language Environment environment was initialized using ILBOSTP0
- PL/I considerations: For PL/I MTF applications, Language Environment provides AMODE switching. Therefore, the first routine of a task can be in AMODE 24.
- 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).
- XPLINK considerations: 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 is forced to ON. No AMODE 24 routines are allowed in an enclave that uses XPLINK. No message is issued to indicate this action. If a Language Environment runtime options report is generated using the RPTOPTS runtime option, the ALL31 option is reported as "Override" under the LAST WHERE SET column.
- **Guideline:** ALL31 should 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.

Performance consideration

If your application consists entirely of AMODE 31 routines, it will run faster and use less below-the-line storage with ALL31(ON) than with ALL31(OFF).

For more information

- See "STACK" on page 75 for information about the STACK runtime option.
- See "XPLINK" on page 101 for information about the XPLINK runtime option.

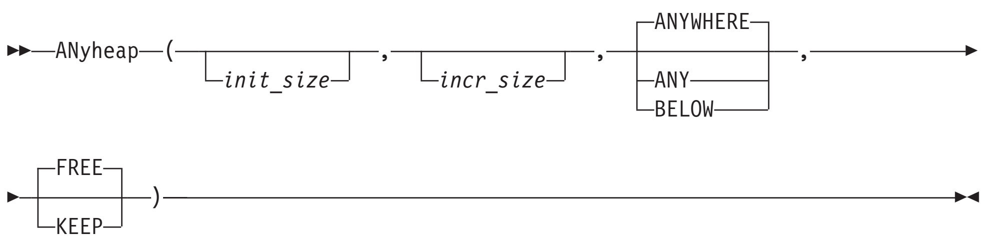
ANYHEAP

The ANYHEAP runtime option controls the allocation of unrestricted library heap storage (anywhere heap). Storage that is unrestricted can be located anywhere in 31-bit addressable storage.

The default value for non-CICS applications is ANYHEAP(16K,8K,ANYWHERE,FREE).

The default value for CICS applications is ANYHEAP(4K,4080,ANYWHERE,FREE).

Syntax



`init_size`
: Determines the initial size of the anywhere heap. This value can be specified as *n*, *n*K, or *n*M 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. This value can be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

ANYWHERE|ANY
: Specifies that anywhere heap can be allocated anywhere in 31–bit addressable storage. If there is no storage available above the 16M line, storage is acquired below the 16-MB line.

BELOW
: Specifies that anywhere heap is allocated below the 16M line.

FREE
: Specifies that an anywhere heap increment is released when the last of the storage within that increment is freed.

KEEP
: Specifies that an anywhere heap increment is **not** released when the last of the storage within that increment is freed.

CICS consideration

- If ANYHEAP(0) is specified, the initial anywhere heap segment is obtained on the first use and will be based on the increment size.
- The maximum initial and increment size for the anywhere heap is 1 gigabyte (1024M).
- The default increment size 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 16 MB line). If you choose to change the increment size, it is recommended that you adjust for the 16 byte CICS storage check zone.

z/OS UNIX considerations

- In a multithreaded environment, the anywhere heap is shared by all threads in the process.
- When ALL31(ON) is in effect, Language Environment allocates thread-specific control blocks from the anywhere heap.

Performance consideration

You can improve performance with the ANYHEAP runtime option by specifying values that minimize the number of times the operating system allocates storage. See “RPTSTG” on page 71 for information about how to generate a report you can use to determine the optimum values for the ANYHEAP runtime option.

For more information

For more information about heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.

ARGPARSE | NOARGPARSE (C only)

Derivation: ARGument PARSE

ARGPARSE specifies if command-line arguments to a C/C++ program are to be parsed. This option does not apply to non-C/C++ languages and can be specified with the `#pragma runopts` directive or the ARGPARSE or NOARGPARSE compiler option.

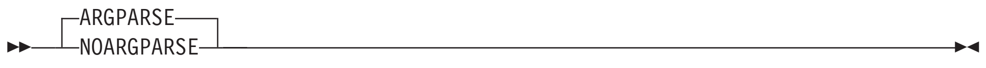
Restriction: You cannot set this option at the system level, region level, or in the CEEBXITA assembler user exit interface

The default value for non-CICS applications is ARGPARSE.

ARGPARSE is ignored under CICS.

The default value for AMODE 64 applications is ARGPARSE.

Syntax



<u>**ARGPARSE**</u>
: Specifies that arguments given on the command line are to be parsed and given to the `main()` function in the usual C argument format (`argv`, and `argc`).

NOARGPARSE
: Specifies that arguments given on the command line are not parsed but are passed to the `main()` function as one string. Therefore, `argc` has a value of 2, and `argv[1]` contains a pointer to the command-line string.

Note: NOARGPARSE is ignored for programs that utilize spawn() or exec() or for any program that is started by the z/OS UNIX System Services shell or by the BPXBATCH utility.

Usage notes

You must specify ARGPARSE for the REDIR runtime option to have an effect.

For more information

See "REDIR | NOREDIR (C only)" on page 69 for a description of REDIR.

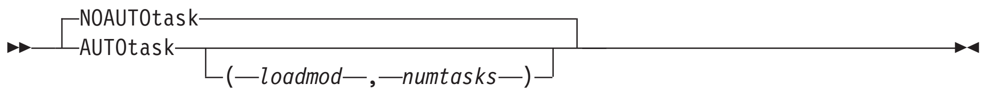
AUTOTASK | NOAUTOTASK (Fortran only)

AUTOTASK specifies if Fortran multitasking facility (MTF) is to be used by your program and the number of tasks that are allowed to be active.

The default value for non-CICS applications is NOAUTOTASK.

AUTOTASK is ignored under CICS.

Syntax



<u>**NOAUTOTASK**</u>
: 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.

BELOWHEAP

The BELOWHEAP runtime option controls the allocation of restricted library heap storage (below heap). Storage that is restricted must be located below the 16M line (24-bit addressable storage).

The default value for non-CICS applications is BELOWHEAP(8K,4K,FREE).

The default value for CICS applications is BELOWHEAP(4K,4080,FREE).

Syntax

```
                                                   ┌─FREE─┐
►►──BElowheap──(──┬───────────┬──,──┬───────────┬──,──┼──────┼──)──────────►◄
                  └─init_size─┘     └─incr_size─┘     └─KEEP─┘
```

init_size
: Determines the minimum initial size of the below heap storage. This value can be specified as *n*, *n*K, or *n*M 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 16M line, and is specified in *n*, *n*K, or *n*M 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 within that increment is freed.

CICS consideration

The default increment size 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 16-MB line). If you choose to change the increment size, it is recommended that you adjust for the 16-byte CICS storage check zone.

z/OS UNIX considerations

- In a multithreaded environment, the below heap is shared by all threads in the process.
- When ALL31(OFF) is in effect, Language Environment allocates thread-specific control blocks from the below heap.

Usage notes

If BELOWHEAP(0) is specified, the initial below heap segment is obtained on the first use and is based on the increment size.

Performance consideration

You can improve performance with the BELOWHEAP runtime option by specifying values that minimize the number of times the operating system allocates storage. See “RPTSTG” on page 71 for information about how to generate a report you can use to determine the optimum values for the BELOWHEAP runtime option.

For more information

For more information about heap storage and heap storage tuning 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 if runtime options or program arguments appear first in the parameter string. You can only specify this option at the system level, region level, or in a CEEUOPT.

CBLPSHPOP (COBOL only)

Derivation: COBOL PUSH POP

CBLPSHPOP controls if CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL subroutine is called. CBLPSHPOP is ignored in non-CICS environments.

Tip: 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 runtime option on an enclave basis using CEEUOPT.

There is no default value for non-CICS applications because CBLPSHPOP is ignored.

The default value for CICS applications is CBLPSHPOP(ON).

Syntax

```
                     ┌─ON──┐
►►──CBLPshpop──(─────┼─────┼──)─────────────────────────────────────►◄
                     └─OFF─┘
```

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.

Performance consideration

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*.
- To determine where to locate information about CICS commands, see *CICS Master Index*.

CBLQDA (COBOL only)

Derivation: COBOL QSAM Dynamic Allocation

CBLQDA controls COBOL QSAM dynamic allocation on an OPEN statement.

The default value for non-CICS applications is CBLQDA(OFF).

CBLQDA is ignored under CICS.

Syntax

```
                 ┌─OFF─┐
►►──CBLQda──(────┴─────┴──)────────────────────────────────────────►◄
                 └─ON──┘
```

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.

Usage notes

- CBLQDA(OFF) is the recommended default because it 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.
- CBLQDA does not affect dynamic allocation for the message file specified in MSGFILE or the Language Environment formatted dump file (CEEDUMP) .

CEEDUMP

Derivation: Common Execution Environment DUMP

CEEDUMP

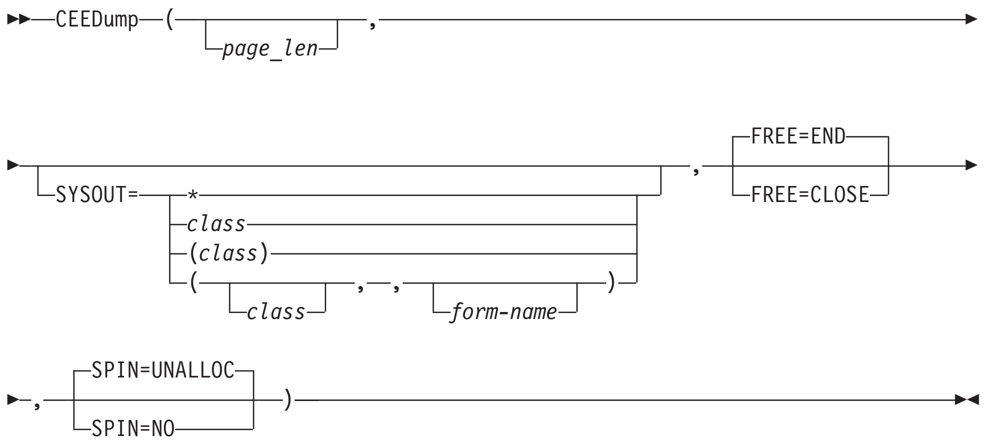
The CEEDUMP runtime option is used to specify options to control the processing of the Language Environment dump report.

The default value for non-CICS applications is CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC).

The default value for CICS applications is CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC).

The default value for AMODE 64 is CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC).

Syntax



`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 should contain 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, but the second parameter should always be omitted:.

  `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 specified fails, SYSOUT=* is set and message CEE3785I is issued.

  `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 will have one of the following JCL DD attributes:

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

CLOSE
The FREE=CLOSE DD attribute requests that the system unallocates 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 will have one of the following attributes:

UNALLOC
The SPIN=UNALLOC attribute indicates the system should 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 attribute indicates the system should 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.

z/OS UNIX considerations

The SYSOUT=, FREE= and SPIN= suboptions do not have any effect on a CEEDUMP report taken in a z/OS UNIX file system.

CICS consideration

The SYSOUT=, FREE= and SPIN= suboptions do not have any effect on a CEEDUMP report taken under CICS.

Usage notes

- If a CEEDUMP DD card is explicitly coded in a job step, Language Environment ignores any SYSOUT class, FREE, SPIN or *form-name* 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 Customization*.
- Language Environment supports the use of a CEEDUMP DDNAME dynamically allocated with the XTIOT, UCB nocapture, or DSAB-above-the-line options specified in the SVC99 parameters (S99TIOEX, S99ACUCB, S99DSABA flags).

CICS considerations

- CEED(,SYSOUT=X,FREE=CLOSE)

This dynamically allocates a CEEDUMP with SYSOUT class X and the FREE=CLOSE DD attribute. The other implicit suboptions are *page_len* and SPIN, which default to whatever values are currently set in the CEEDUMP runtime option.

- CEEDUMP(40)

  The CEEDUMP report contains a page length of 40 lines. The other implicit suboptions are SYSOUT=, FREE=, and SPIN=, which default to whatever values are currently set in the CEEDUMP runtime option.

CHECK (COBOL only)

If your COBOL application was compiled with the SSRANGE compile option, specifying the CHECK(ON) runtime option enables range checking of each index, subscript, and reference modification.

The default value for non-CICS applications is CHECK(ON).

The default value for CICS applications is CHECK(ON).

Syntax

```
                 ┌─ON──┐
►►──CHeck──(─────┼─────┼──)──────────────────────────────────────────────►◄
                 └─OFF─┘
```

ON Specifies that runtime range checking is performed.

OFF
Specifies that runtime range checking is not performed.

Usage notes

CHECK(ON) has no effect if the NOSSRANGE COBOL compile option was specified.

Performance consideration

If your COBOL program was compiled with the SSRANGE compile option, and you are not testing or debugging an application, performance improves when you specify CHECK(OFF). For COBOL 4.2 and earlier you can disable the SSRANGE code with CHECK(OFF). For COBOL 5.1 and later, you cannot disable it.

COUNTRY

Specifying a `country_code` with the COUNTRY runtime option determines:
- Date and time formats
- The currency symbol
- The decimal separator
- The thousands separator

COUNTRY affects only the Language Environment NLS services, not the Language Environment locale callable services.

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

The default value for non-CICS applications is COUNTRY(US) .

The default value for CICS applications is COUNTRY(US) .

Syntax

```
►►──COUNTRy──(──┬──────────────┬──)──────────────────────────────►◄
                └─country_code─┘
```

country_code
: A 2-character code that indicates to Language Environment the country on which to base the default settings. Table 32 on page 481 contains a list of valid country codes.

Usage notes

- If you specify an unsupported *country_code* , Language Environment accepts the value and issues an informational message.

  If an unsupported *country_code* is specified on the COUNTRY runtime option and one of the services listed in "National Language Support callable services" on page 112 is called with the optional *country_code* parameter omitted, the called service returns its specified default value.
- 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. The COUNTRY setting does not affect these locale-sensitive functions and locale callable services.

  To change the locale, you can use the `setlocale()` library function or the CEESETL callable service. These affect only C/C++ locale-sensitive functions and Language Environment locale callable services, not the COUNTRY runtime option.

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

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

For more information

- For more information about the CEE3CTY callable service, see "CEE3CTY—Set default country" on page 133.
- For more information about the RPTOPTS runtime option, see "RPTOPTS" on page 69.
- For more information about the RPTSTG runtime option, see "RPTSTG" on page 71.
- For a list of countries and their codes, see Appendix A, "IBM-supplied country code defaults," on page 481.

- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 411.
- For more information about `setlocale()`, see *z/OS XL C/C++ Programming Guide*.

DEBUG (COBOL only)

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

The default value for non-CICS applications is NODEBUG.

The default value for CICS applications is NODEBUG.

Syntax

```
      ┌─NODEBUG─┐
►►────┴─DEBUG───┴──────────────────────────────────────────────►◄
```

NODEBUG
: Suppresses the COBOL batch debugging features.

DEBUG
: 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.

Usage notes

- For information about specifying this option at the system or region level, see *z/OS Language Environment Customization*.
- For information about specifying this option in CEEUOPT, see *z/OS Language Environment Programming Guide*.
- Use DEBUG and NODEBUG only on the command line.

Performance consideration

Because DEBUG gives worse runtime performance than NODEBUG, you should use it only during application development or debugging.

For more information

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) for more information about the USE FOR DEBUGGING declarative.

DEPTHCONDLMT

Derivation: DEPTH of nested CONDition LiMiT

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.

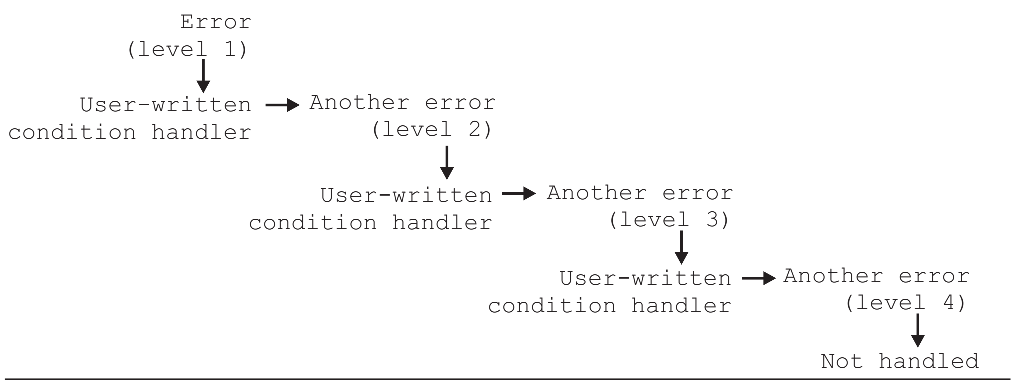


Figure 1. Effect of DEPTHCONDLMT(3) on condition handling

The default value for non-CICS applications is DEPTHCONDLMT(10).

The default value for CICS applications is DEPTHCONDLMT(10).

Syntax

```
►►──DEPthcondlmt──(─────────)──────────────────────────────────►◄
                     └─limit─┘
```

`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 *limit* value of 1 specifies handling of the initial condition, but does not allow handling of nested conditions that occur while handling a condition. With a *limit* value of 5, for example, the initial condition and four nested conditions are processed. In this case, there can be no further nesting of conditions.

  If the number of nested conditions exceeds the *limit*, the application ends with abend 4087.

z/OS UNIX considerations

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 consideration: DEPTHCONDLMT(0) provides PL/I compatibility.
- PL/I MTF consideration: 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 ends 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 if 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.

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

The non-CICS default value is DYNDUMP(**userid*,NODYNAMIC,TDUMP).

DYNDUMP is ignored under CICS.

The AMODE 64 default value is DYNDUMP(**userid*,NODYNAMIC,TDUMP).

Syntax

```
              ┌─*USERID───┐     ┌─NODYNAMIC─┐    ┌─TDUMP───┐
►►──DYNdump──(──┼─hlq───────┼──,──┼─DYNAMIC───┼──,──┴─NOTDUMP─┴──)──────────►◄
              └─*TSOPREFIX┘     ├─FORCE─────┤
                                └─BOTH──────┘
```

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

Each keyword may be followed by additional characters to be used to create the dataset name. When appended to the USERID or the TSO prefix, they form the hlq used when creating the dump data set.

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.Thhmmsst.jobname`
- When the application is exec()ed and not multilevel security:
  `hlq.Djjj.Thhmmsst.Ppppppppp`
- When the application is multilevel security and not exec()ed:
  `hlq.MLS-level.Djjj.Thhmmsst.jobname`
- When the application is both multilevel security and exec()ed:
  `hlq.MLS-level.Djjj.Thhmmsst.Ppppppppp`

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.

Usage notes

- **Recommendations:**
  - 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 is 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.

CICS considerations

- DYNDUMP(smith, force, notdump)

  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)

  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)

  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)

  A dynamic dump for a U4039 ABEND is taken to SMITH.D109.T234512.JOBZ3.
- DYNDUMP(*USERID.HOT.DUMPS,NODYNAMIC,TDUMP)
- DYNDUMP(*TSOPRE.A1234567.B1234567,NODYNAMIC,TDUMP)

ENV (C only)

Derivation: ENVironment

ENV specifies the operating system for your C application. z/OS XL C++ users can get the same behavior by using the z/OS XL C++ compiler option TARGET(IMS™) to specify ENV(IMS).

Restriction: This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive. You cannot set this option at the system level, region level, or in the CEEBXITA assembler user exit interface.

For non-CICS applications, the ENV option differs from other runtime options that it does not have a standard default. The default depends on the system (CMS, IMS, or MVS) in which compilation occurs.

ENV is ignored under CICS.

Syntax

```
►►──ENV──(──┬─CMS─┬──)──────────────────────────────────────────►◄
            ├─IMS─┤
            └─MVS─┘
```

CMS
Specifies that the C application runs in a CMS environment.

IMS
Specifies that the C application runs in an IMS environment. You do not need to specify the ENV option if your application is invoked under IMS but does not actually use IMS facilities.

MVS
Specifies that the C application runs in an MVS environment.

z/OS UNIX considerations

- In a multithreaded environment, the ENV option is shared by all threads in the process.

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 runtime 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 runtime option on the command level.

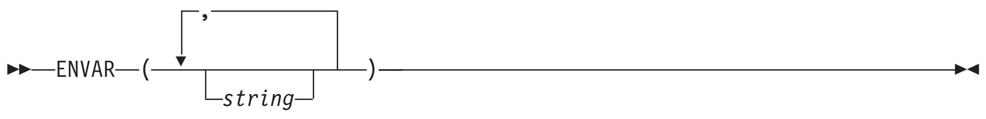
When you specify the RPTOPTS runtime option, the output for the ENVAR runtime option contains a separate entry for each source where ENVAR was specified with the environment variables from that source.

The default value for non-CICS applications is ENVAR('').

The default value for CICS applications is ENVAR('').

The default value for AMODE 64 applications is ENVAR('').

Syntax



`string`

Is specified as *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 *string* in either single or double quotation marks to distinguish it from other strings. You can specify multiple environment variables, separating the *name=value* pairs with commas. Quotation marks are required when specifying multiple variables.

z/OS UNIX considerations

In a multithreaded environment, the environment variables are shared by all threads in the process.

Usage notes

- The entire value of the ENVAR operand, whether a single string or multiple strings, cannot exceed 250 characters.
- *string* cannot contain DBCS characters.
- If ENVAR is specified as part of the _CEE_RUNOPTS environment variable, it is ignored.
- The ENVAR option functions independently of the POSIX runtime option setting.
- C/C++ consideration—An application can access the environment variables using C function `getenv()` or the POSIX variable *environ*, which is defined as:

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

  Guideline: You should access environment variables through the `getenv()` function, especially in a multithread environment. getenv() serializes access to the environment variables.

  Environment variables that are passed to the exec or spawn family of functions replace those established by the ENVAR option in the new environment.

For more information

- For more information about the RPTOPTS runtime option, see "RPTOPTS" on page 69.
- For more information about `getenv()`, `putenv()`, `setenv()`, `clearenv()`, `system()`, and the exec and spawn family of functions, see *z/OS XL C/C++ Runtime Library Reference* .
- For more information about the order of precedence of runtime option sources, see *z/OS Language Environment Programming Guide*.

ERRCOUNT

Derivation: ERRor COUNTer

ERRCOUNT specifies how many conditions of severity 2, 3, or 4 can occur per thread before the enclave terminates abnormally. When the number specified in ERRCOUNT is exceeded, Language Environment ends with ABEND U4091 RC=B.

The default value for non-CICS applications is ERRCOUNT(0).

The default value for CICS applications is ERRCOUNT(0).

Syntax

```
►►──ERrcount──(──┬────────┬──)─────────────────────────────────────►◄
                 └─number─┘
```

number
An integer of 0 or greater value that specifies the number of severity 2, 3, and 4 conditions allowed for each thread. An unlimited number of conditions is allowed if you specify 0. If the number of conditions exceeds the limit, the application ends with abend 4091 RC=B.

z/OS UNIX considerations

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

Usage notes

- Language Environment does not count severity 0 or 1 conditions toward ERRCOUNT.
- ERRCOUNT only applies when conditions are handled by a user condition handler, signal catcher, PL/I on-unit, or a language-specific condition handler. Any unhandled condition of severity 2, 3, or 4 causes the enclave to terminate.
- COBOL consideration: The COBOL runtime library separately counts its severity 1 (warning) messages. When the limit of 256 IGZnnnnW messages is reached, the COBOL runtime library will issue message IGZ0041W, which indicates that the limit of warning messages has been reached. Any further COBOL warning messages are suppressed and processing continues.
- PL/I consideration: You should use ERRCOUNT(0) in a PL/I environment to avoid unexpected termination of your application. Some conditions, such as ENDPAGE, can occur many times in an application.
- PL/I MTF consideration: 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.
- C++ consideration: C++ throw does not affect ERRCOUNT.

For more information

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

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.

The default value for non-CICS applications is ERRUNIT(6).

ERRUNIT is ignored under CICS.

Syntax

```
►►──ERRUnit──(──────────)──────────────────────────────────────────►◄
              └─number─┘
```

number
A number in the range 0-99.

Usage notes

The Language Environment message file and the file connected to the Fortran error message unit are the same.

EXECOPS | NOEXECOPS (C only)

EXECOPS specifies if you can enter runtime options on the command line.

Restriction: This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive. You cannot set this option at the system level, region level, or in the CEEBXITA assembler user exit interface.

The default value for non-CICS applications is EXECOPS.

This option is ignored under CICS.

The default value for AMODE 64 applications is EXECOPS.

Syntax

```
    ┌─EXECOPS───┐
►►──┴─NOEXECOPS─┴──────────────────────────────────────────────────►◄
```

EXECOPS
Specifies that you can enter runtime options on the command line. Language Environment parses the argument string to separate runtime options from application arguments. The runtime options are interpreted by Language Environment and the application arguments are passed to the application.

NOEXECOPS
Specifies that you cannot enter runtime options on the command line. Language Environment passes the entire argument string to the application.

For more information

For more information about the format of the argument string and how it is parsed, see *z/OS Language Environment Programming Guide*.

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 run time. This option is intended for use with applications called by Fortran that reallocate Fortran's supplied DD names.

The default value for non-CICS applications is FILEHIST.

FILEHIST is ignored under CICS

Syntax

```
      ┌─FIlehist───┐
►►────┴─NOFIlehist─┴──────────────────────────────────────────────────►◄
```

FILEHIST
Causes the history of a file to be used in determining its existence. It checks to see if:
- 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).

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 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 reopen an empty file that has been closed with STATUS='KEEP', because the file does not appear to exist to Fortran.

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 z/OS UNIX file system files open with fopen() or freopen(), or upon the first I/O to a pipe created with popen(). You should be familiar with the concept of file tagging, automatic conversion, and the program and file CCSIDs to use the runtime option properly. See *z/OS XL C/C++ Programming Guide* for more information.

The default value for non-CICS applications is FILETAG(NOAUTOCVT,NOAUTOTAG).

FILETAG is ignored under CICS.

The default value for AMODE 64 applications is FILETAG(NOAUTOCVT,NOAUTOTAG).

Syntax

```
           ┌─NOAUTOCVT─┐   ┌─NOAUTOTAG─┐
▶▶─FILETAG─(─┼───────────┼─,─┼───────────┼─)────────────────────▶◀
           └─AUTOCVT───┘   └─AUTOTAG───┘
```

NOAUTOCVT
Disables automatic text conversion.

AUTOCVT
Enables automatic text conversion for untagged UNIX file system files opened using `fopen()` or `freopen()`. The conversion for an untagged file is from the program CCSID to the EBCDIC CCSID as specified by the _BPXK_CCSIDS 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_CCSIDS 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 UNIX file system files applies only to files opened in text mode. An untagged file opened in binary mode is not converted automatically. You can control the tagging 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 UNIX file system files that are automatically tagged by the AUTOTAG suboption. A UNIX file system 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 situations 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 BPXPRM*xx* member on your system.

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

NOAUTOTAG
Disables the automatic tagging of new or empty z/OS UNIX files.

AUTOTAG
Enables automatic file tagging, on the first write, of new or empty z/OS UNIX files opened with `fopen()`, `freopen()`, or `popen()`.

Usage notes

- Avoid the following 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 codepage and 819 indicates the ASCII ISO8859-1 codepage.
- 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 if the redirected streams are tagged. See *z/OS UNIX System Services Command Reference* for more information.
- Automatic tagging for a z/OS UNIX file is done by the system at first write. 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, `fopoen()` or `freopen()` of a z/OS UNIX file fails if it cannot determine if the file exists or if it cannot determine the size.

FLOW (COBOL only)

FLOW controls the FLOW output produced by OS/VS COBOL programs. You cannot set this option at the system or region levels.

The default value for non-CICS applications is NOFLOW.

The default value for CICS applications is NOFLOW.

Syntax

```
     ┌─NOFLOW──┐
►►───┼─FLOW────┼──────────────────────────────────────────────────►◄
     ├─FLOW(n)─┤
     ├─FLOW=n──┤
     └─FLOWn───┘
```

NOFLOW
: Suppresses the OS/VS COBOL FLOW output.

FLOW
Allows the OS/VS COBOL FLOW output.

n Specifies the number of procedures traced. The number can be any integer from 1 to 99, inclusive.

HEAP

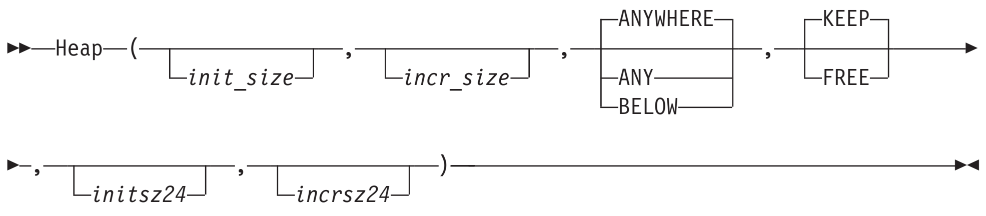
HEAP controls the allocation of user heap storage and specifies how that storage is managed. Heaps are storage areas containing user-controlled dynamically allocated variables or data. Examples of these are:

- C data 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
- Data allocated as a result of a call to the CEEGTST Language Environment callable service

The default value for non-CICS applications is HEAP(32K,32K,ANYWHERE,KEEP,8K,4K).

The default value for CICS applications is HEAP(4K,4080,ANYWHERE,KEEP,4K,4080).

Syntax



init_size
Determines the initial allocation of heap storage. This value can be specified as *n*, *n*K, or *n*M bytes of storage. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. 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 user heap storage. This value can be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

ANYWHERE|ANY
Specifies that user 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.

BELOW
: Specifies that user heap storage is allocated below the 16M line in storage.

  Restriction: The HEAPPOOLS option is ignored when the BELOW suboption is specified.

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.

initsz24
: Determines the minimum initial size of user heap storage that is obtained below the 16M line for AMODE 24 applications that specify ANYWHERE in the HEAP runtime option. This value can be specified as *n*, *n*K, or *n*M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

incrsz24
: Determines the minimum size of any subsequent increment to user heap storage that is obtained below the 16M line for AMODE 24 applications that specify ANYWHERE in the HEAP runtime option. This value can be specified as *n*, *n*K, or *n*M number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

CICS consideration

- If HEAP(,,ANYWHERE,,,) is in effect, the maximum size of a heap segment is 1 gigabyte (1024 MB).
- 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 when the storage allocation is below the 16 MB line.

  Guideline: If you choose to change the increment size, you should adjust for the 16 byte CICS storage check zone.

z/OS UNIX considerations

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

Usage notes

- Applications running in AMODE 24 that request heap storage get the storage below the 16M line regardless of the setting of ANYWHERE | BELOW.
- COBOL consideration—You can use the HEAP option to provide function similar to the VS COBOL II space management tuning table.
- PL/I consideration—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 runs in AMODE 31.
- PL/I MTF consideration—In a PL/I MTF application, HEAP specifies the heap storage allocation and management for a PL/I main task.

Performance consideration

You can improve performance with the HEAP runtime option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 71 for information about how to generate a report you can use to determine the optimum values for the HEAP runtime option.

For more information

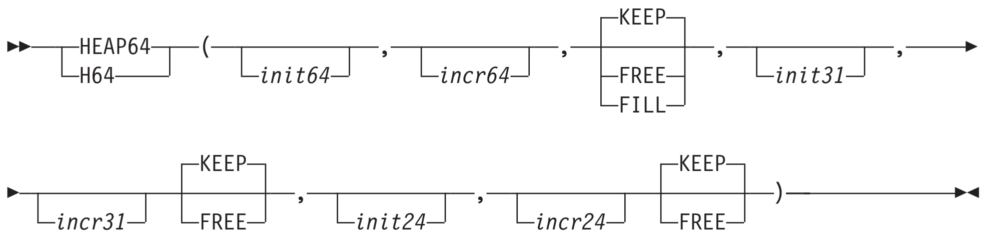
- For more information about Language Environment heap storage, see *z/OS Language Environment Programming Guide*.
- For more information about the CEECRHP callable service, see "CEECRHP—Create new additional heap" on page 221.
- For more information about the CEEGTST callable service, see "CEEGTST—Get heap storage" on page 313.
- For more information about the RPTSTG runtime option, see "RPTSTG" on page 71.

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 containing user-controlled dynamically allocated variables or data. An example is C data allocated as a result of the `malloc()`, `calloc()`, and `realloc()` functions.

The default value for AMODE 64 applications is HEAP64(1M,1M,KEEP,32K,32K,KEEP,4K,4K,FREE).

Syntax



init64
Determines the initial allocation of heap storage that is obtained above the 2G bar. Specify this value as *n*M 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 *n*M 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 16M line and below the 2G bar. This value can be specified as *n*, *n*K, or *n*M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. 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 16M line and below the 2G bar. This value can be specified as *n*, *n*K, or *n*M number of bytes. 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 16M line. This value can be specified as *n*, *n*K, or *n*M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. 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 16M line. This value can be specified as *n*, *n*K, or *n*M number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

z/OS UNIX considerations

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

Performance consideration

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 71 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 heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.
- For more information about the RPTSTG runtime option, see “RPTSTG” on page 71.

HEAPCHK

Derivation: HEAP storage CHecKing

HEAPCHK performs diagnostic tests against the user heap.

The default value for non-CICS applications is HEAPCHK(OFF,1,0,0,0,1024,0,1024,0).

The default value for CICS applications is HEAPCHK(OFF,1,0,0,0,1024,0,1024,0).

The default value for AMODE 64 applications is HEAPCHK(OFF,1,0,0,0,1024,0,1024,0).

Syntax

```
            ┌─OFF─┐
►►──HEAPChk──(──┤     ├──,────────────────────────────────────────►
            └─ON──┘

►──┬───────────┬──,──┬───────┬──,──┬────────────┬──,──┬─────────────────┬──►
   └─frequency─┘     └─delay─┘     └─call depth─┘     └─pool call depth─┘

►──┬────────────────┬──,──┬─────────────┬──,──────────────────────►
   └─num of entries─┘     └─pool number─┘

►──┬───────────────────┬──,──┬────────────────┬──)────────────────►◄
   └─num of entries 31─┘     └─pool number 31─┘
```

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*, *n*K or *n*M. 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 heap to be checked at every nth 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 nth 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. Use a value of 10 to ensure an adequate call depth is displayed so that you can identify the storage leak.

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. Use a value of 10 to ensure an adequate call depth is displayed so that you can identify the storage leak.

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 AMODE 64 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 AMODE 64 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.

Usage notes

- When user heap damage is detected, Language Environment immediately issues an ABEND U4042 RC=0. To obtain a system dump of this abend, either specify TDUMP for the third suboption of the DYNDUMP runtime option or ensure that a SYSMDUMP DD is available.
- If HEAPCHK(ON) is used with STORAGE(,heap_free_value), all free areas of the heap are also checked.
- 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 (HEAPCHK(ON,0,0,10,0)).
- Under normal termination conditions, if the call depth is greater than zero, the heap storage diagnostics report is written to the CEEDUMP report, independent of the TERMTHDACT setting.
- You can also generate a heap storage diagnostics report by calling CEE3DMP with the BLOCKS option.

- **Guideline:** Since HEAPCHK does not validate individual cells within a cell pool, you should specify HEAPPOOLS(OFF) when running HEAPCHK to diagnose storage overlays in the heap.
- 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 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.
- For AMODE 64 applications, *number of entries* and *pool number* control tracing for the set of heap pools located in storage above the 2-GB bar. *Number of entries 31* and *pool number 31* control tracing for the set of heap pools located in storage above the 16-MB line and below the 2-GB bar. *Pool call depth* applies to both sets of heap pools.
- There is no interaction between specifying the HEAPZONES and HEAPCHK runtime options, but setting both at the same time is not recommended by IBM.

Performance consideration

Specifying HEAPCHK(ON) can result in a performance degradation due to the user heap diagnostic testing that is performed.

For more information

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

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

Derivation: HEAP storage POOLS

The HEAPPOOLS runtime option is used to control an optional user heap storage management algorithm, which is known as *heap pools*. This algorithm is designed to improve the performance of multithreaded applications with a high frequency of calls to malloc(), __malloc31(), calloc(), realloc(), free(), and operators `new` and `delete`. When active, heap pools virtually eliminate contention for user heap storage.

HEAPPOOLS runtime option can be used by AMODE 64 applications to manage user heap storage above the 16 M line and below the 2 G bar.

The default value for non-CICS applications is 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).

The default value for CICS applications is 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).

The default value for AMODE 64 applications is 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).

Syntax

```
                                ┌─* repeats 11 times─────────────────────────────┐
                                ▼                                                │
►►──HEAPPools──(──┬─OFF───┬──,──┬─cell-size──────────────────┬─, percentage──┴──)────────►◄
                  ├─ON────┤     └─(cell-size, pool-count)────┘
                  └─ALIGN─┘
```

OFF
: Specifies that the heap pools algorithm is not used.

ON Specifies that the heap pools algorithm is used.

ALIGN
: Specifies that Language Environment will structure 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
: Specifies the size of the cells in a heap pool, which is 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.

percentage
: The size of this heap pool and any of its extents is determined by multiplying this percentage by the *init_size* value that was specified in the HEAP runtime option. The percentage must be in a range from 1 to 90.

Usage notes

- 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 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.
- 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.
- The FREE suboption on the HEAP runtime option has no effect on any heap segment 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. To avoid wasting storage, see the heap pool tuning tips specified in *z/OS Language Environment Programming Guide*.
- The HEAPPOOLS runtime option has no effect when BELOW is specified as the location on the HEAP runtime option.
- Mixing of the storage management AWIs (CEEGTST(), CEEFRST() and CEECZST()) and the C/C++ intrinsic functions (malloc(), calloc(), realloc() and free()) are 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.

- Using the ALIGN suboption might cause an increase in the amount of heap storage that is used by an application.
- You should 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.
- Use of a vendor heap manager (VHM) overrides the use of the HEAPPOOLS runtime option.
- If the RPTSTG runtime option is specified 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.
- The HEAPPOOLS runtime option can be used by AMODE 64 applications to manage user heap storage above the 16M line and below the 2G bar.
- When cell-size is specified without parenthesis, pool-count defaults to 1 rather than being picked up from an earlier setting of pool-count. For example, specifying 128 is treated like specifying (128,1).
- When cell-size is specified with 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.

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.
- 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 could 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.

HEAPPOOLS64 (C/C++ and 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, for AMODE 64 applications. 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.

The default value for AMODE64 applications is 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).

Syntax

```
                                    ┌─* repeats 11 times──────────────────────────┐
                                    ▼                                              │
►►──┬─HEAPPOOLS64─┬──(──┬─OFF───┬──,──┬─cell-size──────────────────┬─, count──┴──)──────►◄
    └─HP64────────┘     ├─ON────┤     └─(cell-size, pool-count)─┘
                        └─ALIGN─┘
```

OFF
Specifies that the heappools algorithm is not being used.

ON Specifies that the heappools algorithm is being used.

ALIGN
Specifies that Language Environment will structure 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.

Usage notes

- 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.
- 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.
- HEAPPOOLS runtime option can be used by AMODE 64 applications to manage user heap storage above the 16M line and below the 2G bar.
- When cell-size is specified without parenthesis, pool-count defaults to 1 rather than being picked up from an earlier setting of pool-count. For example, specifying 128 is treated like specifying (128,1).
- When cell-size is specified with 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.

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 and count.
- 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.

For more information

For more information about heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.

HEAPZONES

Derivation: HEAP check ZONES

The HEAPZONES runtime option is used to turn on overlay toleration and checking for user heaps. When activated, the runtime option affects any obtained storage that can be controlled by the HEAP or HEAP64 runtime options. HEAPZONES also affects storage obtained from a heap pool.

A *heap check zone* is an additional piece of storage that is appended to an allocated element during a storage request. The size of the check zone depends on the size31 and size64 suboptions of HEAPZONES. The check zone can be examined for overlays when the heap element is freed.

The default value for non-CICS applications is HEAPZONES(0,ABEND,0,ABEND).

The default value for CICS applications is HEAPZONES(0,ABEND,0,ABEND).

The default value for AMODE 64 applications is HEAPZONES(0,ABEND,0,ABEND).

Syntax

```
                                ┌─ABEND─┐                      ┌─ABEND─┐
►►──HEAPZones──(─┬────────┬──,──┼───────┼──,─┬────────┬──,──┼───────┼──)────────►◄
                 └─size31─┘     ├─QUIET─┤    └─size-64─┘     ├─QUIET─┤
                                ├─MSG───┤                      ├─MSG───┤
                                └─TRACE─┘                      └─TRACE─┘
```

size31
: Controls the size of the check zone for all user heap storage which is below the 2 G bar. The check zone size is rounded up to the nearest multiple of 8 bytes. The maximum size allowed for a check zone is 1024 bytes. Specifying a value of 0 indicates that no check zone is active.

size64
: Controls the size of the check zone for all user heap storage which is above the 2 G bar. The check zone size is rounded up to the nearest multiple of 8 bytes

with a minimum size of 16 bytes. The maximum size allowed for a check zone is 1024 bytes. Specifying a value of 0 indicates that the check zone is not active.

ABEND
: Specifies that check zones are validated and if an overlay is detected, a U4042 ABEND reason code 3 is issued.

QUIET
: Specifies that a heap check zone is appended to every allocated element, but the check zone is not validated.

MSG
: Specifies that check zones are validated and if an overlay is detected, informational messages are issued. For more information about the output for the HEAPZONES runtime option, see *z/OS Language Environment Debugging Guide* .

TRACE
: Specifies that in addition to informational messages, a CEEDUMP containing only a traceback is produced.

Usage notes

- You cannot set HEAPZONES at the system level, region level, or in the CEEBXITA assembler user exit interface.
- You cannot specify an active HEAPZONES setting and RPTSTG(ON) simultaneously. An active HEAPZONES setting causes Language Environment to override RPTSTG to the value OFF and no storage report is produced.
- There is no interaction between specifying the HEAPZONES and HEAPCHK runtime options, but setting both at the same time is not recommended by IBM.

Performance considerations

Using heap check zones can result in a performance degradation and significant additional storage usage. IBM recommends setting the size of a heap check zone to the smallest amount feasible by the application.

INFOMSGFILTER

Derivation: INFOrmational MeSsaGe FILTER

INFOMSGFILTER allows you to activate a filter that eliminates unwanted informational messages. During normal operations, informational messages are sometimes written to the Language Environment message file. If these messages are routed to your terminal, you need to clear them often. If the messages are saved to a data set, they take up disk space and could make it difficult to browse the output for a specific message.

Informational messages are not limited to Language Environment (CEE) messages. They can also be written, using the CEEMSG interface, by other IBM program products or user-written applications.

The default value for non-CICS applications is INFOMSGFILTER(OFF,,,,).

The default value for CICS applications is INFOMSGFILTER(OFF,,,,).

The default value for AMODE 64 applications is INFOMSGFILTER(OFF,,,,).

Syntax

```
                    ┌─OFF─┐
►►──INFOMsgfilter──(─┼─────┼─,─┬────────────┬─,─┬────────────┬─,──────────►
                    └─ON──┘    └─Foreground─┘   └─Background─┘

►─┬──────┬─)───────────────────────────────────────────────────────────►◄
  └─CICS─┘
```

OFF
: Turns off message filtering for all environments.

ON Turns on the message filtering for the specified environments.

Foreground
: Selecting this keyword causes message filtering to be turned on for the TSO and z/OS UNIX environments.

Background
: Selecting this keyword causes message filtering to be turned on for the MVS Batch environment.

CICS
: Selecting this keyword causes message filtering to be turned on in the CICS environment. This option is ignored for AMODE 64 applications.

INQPCOPN (Fortran only)

Derivation: INQuire the Pre-Connected units that are OPeNed

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

The default for non-CICS applications is INQPCOPN.

INQPCOPN is ignored under CICS.

Syntax

```
    ┌─INQpcopn───┐
►►──┴─NOINQpcopn─┴──────────────────────────────────────────────►◄
```

INQPCOPN
: 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.

NOINQPCOPN
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.

INTERRUPT

INTERRUPT causes attention interrupts to be recognized by Language Environment. When you cause an interrupt, Language Environment can give control to your application or to a debug tool.

The default value for non-CICS applications is INTERRUPT(OFF).

INTERRUPT is ignored under CICS.

Syntax

```
                      ┌─OFF─┐
►►──INTerrupt──(──────┼─────┼──)─────────────────────────────────────►◄
                      └─ON──┘
```

OFF
Specifies that Language Environment does not recognize attention interrupts.

ON Specifies that Language Environment recognizes attention interrupts.

z/OS UNIX considerations

- In a multithreaded application, only one thread in the enclave is affected for a particular attention interrupt.

Usage notes

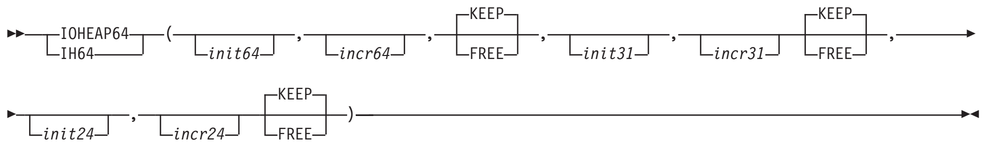
- PL/I consideration: Language Environment supports the PL/I method of polling code. The PL/I routine must be compiled with the INTERRUPT compiler option in order for the INTERRUPT runtime option to have an effect.
- PL/I MTF consideration: To receive the attention interrupt, the PL/I program must be compiled with the INTERRUPT compiler option, and the INTERRUPT runtime option must be in effect.
- PL/I MTF consideration: The INTERRUPT option applies to the enclave. However, only one thread in the enclave is affected for a particular attention interrupt.
- If you have specified the TEST(ERROR) or TEST(ALL) runtime option, the interrupt causes the debug tool to gain control. See "TEST | NOTEST" on page 87 for more information about the TEST runtime option.

IOHEAP64 (AMODE 64 only)

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.

The default value for AMODE 64 applications is IOHEAP64(1M,1M,FREE,12K,8K,FREE,4K,4K,FREE).

Syntax



init64
: Determines the initial allocation of I/O heap storage obtained above the 2G bar. Specify this value as *n*M 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 *n*M 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 16M line and below the 2G bar. This value can be specified as *n*, *n*K, or *n*M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. 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 16M line and below the 2G bar. This value can be specified as *n*, *n*K, or *n*M number of bytes. 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 16M line. This value can be specified as *n*, *n*K, or *n*M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. 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 16M line. This value can be specified as *n*, *n*K, or *n*M number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

Performance consideration

You can improve performance with the IOHEAP64 runtime option by specifying values that minimize the number of times the operating system allocates storage. See “RPTSTG” on page 71 for information about how to generate a report you can use to determine the optimum values for the IOHEAP64 runtime option.

For more information

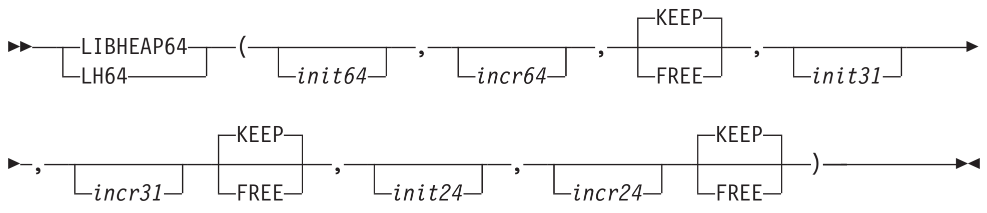
- For more information about heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.
- For more information about the RPTSTG runtime option, see “RPTSTG” on page 71.

LIBHEAP64 (AMODE 64 only)

The LIBHEAP64 runtime option controls the allocation of heap storage that is used by Language Environment for AMODE 64 applications. Storage that is unrestricted can be located anywhere in 64-bit addressable storage.

The default value for AMODE 64 applications is LIBHEAP64(1M,1M,FREE,16K,8K,FREE,8K,4K,FREE).

Syntax



init64
: Determines the initial allocation of library heap storage that is obtained above the 2G bar. Specify this value as *n*M 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 *n*M 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 16M line and below the 2G bar. This value can be specified as *n*, *n*K, or *n*M number of bytes. If 0 is specified, the initial storage is obtained on the

first use and is based on the increment size. 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 16M line and below the 2G bar. This value can be specified as *n*, *n*K, or *n*M number of bytes. 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 16M line. This value can be specified as *n*, *n*K, or *n*M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. 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 16M line. This value can be specified as *n*, *n*K, or *n*M number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

Performance consideration

You can improve performance with the LIBHEAP64 runtime option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 71 for information about how to generate a report you can use to determine the optimum values for the LIBHEAP64 runtime option.

For more information

- For more information about heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.

LIBSTACK

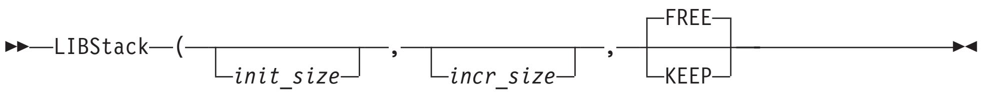
Derivation: LIBrary STACK storage

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

The default value for non-CICS applications is LIBSTACK(4K,4K,FREE).

The default value for CICS applications is LIBSTACK(32,4080,FREE).

Syntax



init_size
Determines the minimum size of the initial library stack storage. This value

can be specified as *n*, *n*K, or *n*M bytes of storage. 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. This value can be specified as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is the larger of 2 values— *incr_size* or the requested size—rounded up to the nearest multiple of 8 bytes. If you specify 0 as *incr_size*, Language Environment gets only the amount of storage that is needed at the time of the request, rounded up to the nearest multiple of 8 bytes.

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 thread ends.

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

CICS considerations

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.

z/OS UNIX considerations

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

Usage notes

Language Environment does not acquire the initial library stack segment until the first program that requires LIBSTACK runs.

Performance considerations

You can improve performance with the LIBSTACK runtime option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 71 for information about how to generate a report you can use to determine the optimum values for the LIBSTACK runtime option.

For more information

- See "RPTSTG" on page 71 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, see *z/OS Language Environment Programming Guide*.

MSGFILE

Derivation: MeSsaGe FILE

MSGFILE specifies the *ddname* and attributes of the data set (message file) where Language Environment directs the following output:

- All Language Environment messages
- Reports generated by the RPTOPTS and RPTSTG runtime options
- Output produced by the CEEMSG and CEEMOUT callable services

The default value for non-CICS applications is MSGFILE(SYSOUT,FBA,121,0,NOENQ).

MSGFILE is ignored under CICS.

Syntax

```
►►──MSGFile──(──┬────────┬──,──┬───────┬──,──┬───────┬──,──┬─────────┬──,────────►
                └─ddname─┘     └─recfm─┘     └─lrecl─┘     └─blksize─┘

   ┌─NOENQ─┐
►──┼───────┼──)──────────────────────────────────────────────────────────────────►◄
   └─ENQ───┘
```

`ddname`
: The ddname of the message file where Language Environment directs the information listed above.

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

`lrecl`
: The record length (LRECL) for the message file. *lrecl* is used when this information is not available either in a data set definition or in the label of an existing data set. *lrecl* is expressed as bytes of storage. The *lrecl* cannot exceed *blksize*. For variable-length record formats, the *lrecl* is limited to *blksize* minus 4.

`blksize`
: The block size (BLKSIZE) for the message file. *blksize* is used when this information is not available either in a data set definition or in the label of an existing data set. *blksize* is expressed as bytes of storage and cannot exceed 32760.

NOENQ
: Specifies that no serialization is performed on *ddname*.

ENQ
: Specifies that serialization is performed on the *ddname* specified in case multiple Language Environment environments are running in the same address space and sharing the same message file.

CICS consideration

- MSGFILE output under CICS is directed to a transient data queue named CESE.

z/OS UNIX considerations

- When multiple threads write to the message file, the output is interwoven by line. To group lines of output, the application must serialize its own output.
- 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 calls fork()/exec(), spawn(), or spawnp(), Language Environment checks if file descriptor 2 (fd2) exists.
  - If fd2 exists, Language Environment uses it.
  - If fd2 does not exist, Language Environment dynamically allocates the message file to the POSIX file system and attempts to open the file SYSOUT in the current working directory. If there is no current directory, SYSOUT is opened in the directory /tmp.

Usage notes

- Under most circumstances, the NOENQ suboption is sufficient and provides better performance. The ENQ suboption is only needed when multiple Language Environment environments are running in the same address space and share the same message file.

  An instance when ENQ might be needed is a batch job that uses ATTACH to create sub-tasks. Each of the sub-tasks is potentially a distinct Language Environment environment, all running with the same default MSGFILE parameters. In this example, each of these environments shares the same message file.

  To avoid conflicts while writing to the shared message file, use the ENQ suboption. Using a different *ddname* for each environment can remove the need to use the ENQ suboption.
- Compiler options, such as the COBOL OUTDD compiler option, can affect if your runtime output goes to MSGFILE *ddname*.
- If there is no *blksize* in the MSGFILE runtime option, in a data set definition, or in the label of an existing data set, the block size is determined as follows:
  - If *recfm* specifies unblocked fixed-length format records (F or FA) or undefined-format records (U or UA), *blksize* is the same as *lrecl*.
  - If *recfm* specifies unblocked variable-length format records (V or VA), *blksize* is *lrecl* plus 4.
  - If *recfm* specifies blocked records (FB, FBA, FBS, FBSA, VB, or VBA) for a DASD device on z/OS, Language Environment uses a *blksize* of 0 so the system can determine the optimum *blksize*.
  - If *recfm* specifies blocked fixed-length format records (FB, FBA, FBS, or FBSA) for a terminal , *blksize* is the same as *lrecl*.
  - If *recfm* specifies blocked variable-length format records (VB or VBA) for a terminal, *blksize* is *lrecl* plus 4.
  - For all other cases, *blksize* is calculated to provide the largest number of records per block, up to 100 records per block, that does not exceed the maximum blksize of 32760.
- Language Environment does not diagnose combinations of *recfm*, *lrecl*, and *blksize* that are not valid but the system can detect an error condition on the first attempt to write to the message file.
- Language Environment does not check the validity of the MSGFILE *ddname*. A *ddname* that is not valid generates an error condition on the first attempt to write to the message file.

- C/C++ consideration—C output directed to `stderr` and `perror()` messages go to the MSGFILE destination.
- PL/I consideration—runtime messages in PL/I programs are directed to the message file instead of to the PL/I SYSPRINT STREAM PRINT file.

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

  Use of MSGFILE(SYSPRINT) restricts the LINESIZE for PL/I programs to a maximum of 255.
- Fortran consideration—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 Fortran runtime Migration Guide.
- Language Environment supports the use of a MSGFILE DDNAME dynamically allocated with the XTIOT, UCB nocapture, or DSAB-above-the-line options specified in the SVC99 parameters (S99TIOEX, S99ACUCB, S99DSABA flags).

For more information

- For more information about the RPTOPTS and RPTSTG runtime options, see "RPTOPTS" on page 69 and "RPTSTG" on page 71.
- For more information about the CEEMSG and CEEMOUT callable services, see "CEEMSG—Get, format, and dispatch a message" on page 373 and "CEEMOUT—Dispatch a message" on page 355.
- 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 examples of getting and formatting messages, including HLL runtime output, see "CEEMSG—Get, format, and dispatch a message" on page 373.
- For more information about `perror()` and `stderr` see the 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 instance specific information (ISI) blocks that Language Environment allocates on a per thread basis for use by the application. The ISI block 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 insufficient number of ISI blocks are available, Language Environment uses the least recently used ISI block.

The default value for non-CICS applications is MSGQ(15).

MSGQ is ignored under CICS.

Syntax

```
►►──MSGQ──(──┬────────┬──)──────────────────────────────────────────►◄
             └─number─┘
```

number
An integer that specifies the number of ISI blocks to maintain on a per thread basis.

Usage notes

- PL/I MTF consideration—In a PL/I MTF application, MSGQ sets the number of message queues allowed for each task.
- The CEECMI callable service allocates storage for ISI blocks if necessary. For information about using the CEECMI callable service, see "CEECMI—Store and load message insert data" on page 216.

For more information

- For more information about the ISI blocks, see *z/OS Language Environment Programming Guide*.

NATLANG

Derivation: NATional LANGuage

NATLANG specifies the initial national language Language Environment uses for the runtime environment, including error messages, month names, and day of the week names. Message translations are provided for Japanese and for uppercase and mixed-case U.S. English. NATLANG also determines how the message facility formats messages.

The default value for non-CICS applications is NATLANG(ENU).

The default value for CICS applications is NATLANG(ENU).

The default value for AMODE 64 applications is applications is NATLANG(ENU).

Syntax

```
               ┌─ENU─┐
►►──NATlang──(─┼─────┼─)────────────────────────────────────────────►◄
               ├─UEN─┤
               └─JPN─┘
```

ENU
A 3-character ID specifying mixed-case U.S. English. Message text consists of SBCS characters and includes both uppercase and lowercase letters.

UEN
A 3-character ID specifying uppercase U.S. 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.

Usage notes

- **Restriction:** CEE3LNG and CEESETL are not available to AMODE 64 applications.
- You can use the CEE3LNG callable service to set the national language.
- If you specify a national language that is not available on your system, Language Environment uses the IBM-supplied default ENU (mixed-case U.S. English).
- Language Environment is sensitive to the national language when it writes storage reports, option reports, and dump output.

  When the national language is uppercase U.S. English or Japanese, the environment variable _CEE_UPPERCASE_DATA can be used to determine if 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, hexadecimal/EBCDIC displays of storage) will be in uppercase.

  When this environment variable is not set or set to a value other than YES, variable data will not be in uppercase. Variable data is never in uppercase when the national language is mixed-case U.S. English.
- Language Environment provides locales used in C/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 the 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 consideration—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 “CEE3LNG—Set national language” on page 172.
- For more information about the CEESETL callable service, see “CEESETL—Set locale operating environment” on page 411.
- 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 if a file is actually deleted based on the STATUS specifier on the OPEN and CLOSE statement, respectively.

The default value for non-CICS applications is OCSTATUS.

OCSTATUS is ignored under CICS.

Syntax

```
      ┌─OCstatus───┐
►►────┴─NOOCstatus─┴───────────────────────────────────────────────────►◄
```

OCSTATUS

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.

NOOCSTATUS

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 NOOCSTATUS. 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.

PAGEFRAMESIZE

Derivation: PAGE FRAME SIZE

PAGEFRAMESIZE specifies the preferred page frame size in virtual storage for HEAP, ANYHEAP, and STACK storage that is obtained during application initialization and runtime.

The default value for non-CICS applications is PAGEFRAMESIZE(4K,4K,4K).

PAGEFRAMESIZE is ignored under CICS.

Syntax

```
                       ┌─4K──────────────┐     ┌─4K─────────────────┐
►►──PAGeframesize──(───┴─────────────────┴──,──┴────────────────────┴──,──────►
                       └─heap_frame_size─┘     └─anyheap_frame_size─┘

   ┌─4K───────────────┐
►──┴──────────────────┴──)──────────────────────────────────────────────────►◄
   └─stack_frame_size─┘
```

`heap_frame_size`
: Specifies the preferred page frame size in virtual storage for initial heap storage allocation and any subsequent heap increments. The page frame size can be specified as one of the following values:

4K Requests the default value of 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

`anyheap_frame_size`
: Specifies the preferred page frame size in virtual storage for initial anywhere heap storage allocation and any subsequent anywhere heap increments. The page frame size can be specified as one of the following values:

4K Requests the default value of 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

`stack_frame_size`
: Specifies the preferred page frame size in virtual storage for initial stack storage allocation and any subsequent stack increments. The page frame size can be specified as one of the following values:

4K Requests the default value of 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

Usage notes

- You cannot set PAGEFRAMESIZE at the system level or region level.
- You cannot specify PAGEFRAMESIZE with the CEEBXITA assembler user exit interface.
- In an XPLINK environment, the *stack_frame_size* suboption only applies to the upward-growing stack.
- If 1-MB page frames are not available, the default 4-KB page frame size will be used. No message is issued to indicate this behavior.
- Page frame sizes larger than 4 KB are not allowed below the 16-MB line. If a PAGEFRAMESIZE parameter specifies 1 MB but that storage type is allocated below the 16-MB line, then the default 4-KB page frames is used. No message is issued to indicate this behavior; however, the runtime options report will show the value that was specified.
- If any PAGEFRAMESIZE parameter specifies 1M, then all of the storage preallocated to the enclave will request 1-MB page frames. The previous two usage notes apply as well.
- By default, THREADSTACK storage comes from the library heap storage that is allocated with the ANYHEAP runtime option. To use 1-MB page frames for the THREADSTACK, ensure the *anyheap_frame_size* suboption specifies 1M.

- When running in a preinitialized environment with an @GETSTORE service routine, a flag is passed to indicate that storage was requested to be backed by 1-MB page frames. For more information about using 1-MB page frames with an @GETSTORE service routine, see *z/OS Language Environment Programming Guide* .

Performance considerations

Large pages are a special-purpose feature to improve performance; therefore, using large pages is not recommended for all types of workloads. For more information about large pages, see the following publications:

- *z/OS MVS Programming: Assembler Services Reference IAR-XCT*
- *z/OS MVS Programming: Authorized Assembler Services Guide*
- *z/OS MVS Programming: Assembler Services Guide*

PAGEFRAMESIZE64

Derivation: PAGE FRAME SIZE for AMODE64

PAGEFRAMESIZE64 specifies the preferred page frame size in virtual storage for HEAP64, LIBHEAP64, IOHEAP64 and STACK64 storage that is obtained during application initialization and runtime.

The default value for AMODE 64 applications is PAGEFRAMESIZE64=(4K,4K,4K,4K,4K,4K,4K)

Syntax

```
                                  ┌─4K──────────────────┐
►►──┬─PAGEFRAMESIZE64─┬──(──┴─heap64_frame_size64─┴──,──────────────►
    └─PAG64───────────┘

   ┌─4K──────────────────┐      ┌─4K─────────────────────┐
►──┴─heap64_frame_size31─┴──,──┴─libheap64_frame_size64─┴──,────────►

   ┌─4K─────────────────────┐      ┌─4K────────────────────┐
►──┴─libheap64_frame_size31─┴──,──┴─ioheap64_frame_size64─┴──,──────►

   ┌─4K────────────────────┐      ┌─4K───────────────┐
►──┴─ioheap64_frame_size31─┴──,──┴─stack64_frame_size─┴──)──────────►◄
```

heap64_frame_size64
: Specifies the preferred page frame size in virtual storage for initial heap storage allocation and any subsequent heap increments above the 2-GB bar. The page frame size can be specified as one of the following values:

  4K Requests the default 4-KB pages.

  1M Requests that 1-MB large pages be used, if available.

heap64_frame_size31
: Specifies the preferred page frame size in virtual storage for initial heap

storage allocation and any subsequent heap increments above the 16-MB line and below the 2-GB bar. The page frame size can be specified as one of the following values:

4K Requests the default 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

libheap64_frame_size64
Specifies the preferred page frame size in virtual storage for initial library heap storage allocation and any subsequent library heap increments above the 2-GB bar. The page frame size can be specified as one of the following values:

4K Requests the default 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

libheap64_frame_size31
Specifies the preferred page frame size in virtual storage for initial library heap storage allocation and any subsequent library heap increments above the 16-MB line and below the 2-GB bar. The page frame size can be specified as one of the following values:

4K Requests the default 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

ioheap64_frame_size64
Specifies the preferred page frame size in virtual storage for initial I/O heap storage allocation and any subsequent I/O heap increments above the 2-GB bar. The page frame size can be specified as one of the following values:

4K Requests the default of 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

ioheap64_frame_size31
Specifies the preferred page frame size in virtual storage for initial I/O heap storage allocation and any subsequent I/O heap increments above the 16-MB line and below the 2-GB bar. The page frame size can be specified as one of the following values:

4K Requests the default 4-KB pages.

1M Requests that 1-MB large pages be used, if available.

stack64_frame_size
Specifies the preferred page frame size in virtual storage for initial stack storage allocation above the 2-GB bar. The page frame size can be specified as one of the following values:

4K Requests the default KB pages.

1M Requests that 1-MB large pages be used, if available.

Usage notes

- You cannot set PAGEFRAMESIZE64 at the system level or region level.
- You cannot specify PAGEFRAMESIZE64 with the CEEBXITA assembler user exit interface.
- If 1-MB page frames are not available, the default 4-KB page frame size is used. No message is issued to indicate this behavior.
- Page frame sizes larger than 4 KB are not allowed below the 16-MB line. If a PAGEFRAMESIZE parameter specifies 1 MB but that storage type is allocated

below the 16-MB line, then the default 4-KB page frames is used. No message is issued to indicate this behavior, and the runtime options report will show the value that was specified.

- If the *heap64_frame_size64* or *hlibheap64_frame_size64* option is specified as 1M, then the initial storage allocation for both will request 1-MB page frames. The previous two usage notes apply as well.
- For THREADSTACK64 storage, the default 4-KB page frame size is used.

Performance considerations

Large pages are a special-purpose feature to improve performance; therefore, using large pages is not recommended for all types of workloads. For more information about large pages, see the following publications:

- *z/OS MVS Programming: Assembler Services Reference IAR-XCT*
- *z/OS MVS Programming: Authorized Assembler Services Guide*
- *z/OS MVS Programming: Assembler Services Guide*

PC (Fortran only)

Derivation: Private Common blocks

PC controls if Fortran status common blocks are shared among load modules.

The default value for non-CICS applications is NOPC.

PC is ignored under CICS.

Syntax

```
      ┌─NOPC─┐
►►────┴─PC───┴──────────────────────────────────────────────────►◄
```

NOPC
: Specifies that Fortran static common blocks with the same name but in different load modules all refer to the same storage. NOPC 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

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

PLIST (C only)

Derivation: Parameter LIST

PLIST

PLIST specifies the format of the invocation parameters your C application receives when you invoke it. Although the CICS, CMS, IMS, MVS, and TSO suboptions of PLIST are supported for compatibility, you should use the HOST or OS suboptions of PLIST. Before using this runtime option, you should review the following restrictions:

- This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive.
- You cannot set this option at the system level, region level, or in the CEEBXITA assembler user exit interface.

The default value for non-CICS applications is PLIST(HOST).

PLIST is ignored under CICS.

Syntax

```
                ┌HOST─┐
►►──PLIST──(────┼CICS─┼──)──────────────────────────────────────────►◄
                ├CMS──┤
                ├IMS──┤
                ├MVS──┤
                ├OS───┤
                └TSO──┘
```

HOST
: The parameter list is a character string. The string is located differently under various systems, as follows:

CMS	If invoked by OSRUN, use the string presented in an MVS-like format located by the pointer held in R1. If not invoked by OSRUN, use the CMS extended parameter list.
TSO	If a command processor parameter list (CPPL) is detected, get the string from the command buffer. If a CPPL is not detected, assume a halfword-prefixed string in the MVS format.
MVS	Use the halfword-prefixed string.

CICS
: The parameter list received by your C application is assumed to be in a CICS format.

CMS
: The parameter list received by your C application is assumed to be in a CMS extended parameter list format.

IMS
: The parameter list received by your C application is assumed to be in an IMS format.

MVS
: The parameter list received by your C application is assumed to be in an MVS format.

OS The parameter list received by your C application is assumed to be in an OS style.

TSO
The parameter list received by your C application is assumed to be in a CPPL format.

Usage notes

- The behavior of C applications with PLIST(HOST) in effect is the same for C++.
- When using the pre-Language Environment-conforming C interface for preinitialization, it is necessary to specify PLIST(MVS) in order to flag preinitialized routines.
- IMS considerations—If your C application runs under IMS, the suboption of PLIST that you specify depends on the version of the C compiler you used. If you compiled your application with Version 2.1 (or earlier) of the C compiler, specify the PLIST(IMS) suboption. If your C++ application runs under IMS, you should specify the z/OS XL C++ compiler option PLIST(OS).
- z/OS UNIX consideration—The PLIST option applies only to the main routine of the initial thread.

PLITASKCOUNT (PL/I only)

Derivation: PL/I TASK COUNTer

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

The default value for non-CICS applications is PLITASKCOUNT(20).

PLITASKCOUNT is ignored under CICS.

Syntax

```
►►──PLITaskcount──(─┬───────┬─)──────────────────────────────►◄
                    └─tasks─┘
```

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.

Usage notes

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

POSIX

Derivation: Portable Operating System Interface - X

POSIX specifies if 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.

The default value for non-CICS applications is POSIX(OFF).

POSIX is ignored under CICS.

The default value for AMODE 64 is POSIX(ON).

Syntax

```
          ┌─OFF─┐
►►──POSix──(─┼─────┼─)──────────────────────────────────────────►◄
          └─ON──┘
```

OFF
Indicates that the application is not POSIX-enabled.

ON Indicates that the application is POSIX-enabled.

Usage notes

- When you set POSIX to ON, you can use functions that are unique to POSIX, such as `pthread_create()`.
- POSIX(ON) applies to z/OS but explicitly excludes CICS. If 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. You can specify POSIX(ON) for both DB2® and IMS applications.
- 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.
- 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 option setting indicates the POSIX semantics to follow.
- 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 a second nested enclave tries to specify the runtime option POSIX(ON) within one Language Environment process, Language Environment ends with abend U4093, reason code 172.

For more information

For more information about POSIX functions that have a kernel dependency or a POSIX ON dependency, see *z/OS XL C/C++ Runtime Library Reference*.

PROFILE

PROFILE controls the use of an optional PROFILER to collect performance data for the running application.

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

The default value for non-CICS applications is PROFILE(OFF,'').

The default value for CICS applications is PROFILE(OFF,'').

The default value for AMODE 64 applications is PROFILE(OFF,'').

Syntax

```
                 ┌─OFF─┐
►►──PROFile──(──┴─────┴──,──┬────────┬──)──────────────────────────►◄
                 └─ON──┘      └─string─┘
```

OFF Indicates that the profile facility is inactive.

ON Indicates that the profile facility is active.

string Profile options that Language Environment passes 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 from a compiler directive.

For more information

For details on the Performance Analyzer, see *Getting Started with C/C++ Productivity Tools for OS/390*, GC09-2918-00.

PRTUNIT (Fortran only)

Derivation: PRinT UNIT

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

The default value for non-CICS applications is PRTUNIT(6).

PRTUNIT is ignored under CICS.

Syntax

```
►►──PRTunit──(──┬────────┬──)──────────────────────────────────────►◄
                 └─number─┘
```

number
A valid unit number in the range 0-99.

PUNUNIT (Fortran only)

Derivation: PUNch UNIT

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

The default value for non-CICS applications is PUNUNIT(7).

PUNUNIT is ignored under CICS.

Syntax

```
►►──PUNunit──(──┬────────┬──)──────────────────────────────────────────►◄
                └─number─┘
```

number
A valid unit number in the range 0-99.

RDRUNIT (Fortran only)

Derivation: ReaDeR UNIT

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

The default value for non-CICS applications is RDRUNIT(5).

RDRUNIT is ignored under CICS.

Syntax

```
►►──RDRunit──(──┬────────┬──)──────────────────────────────────────────►◄
                └─number─┘
```

number
A valid unit number in the range 0-99.

RECPAD (Fortran only)

Derivation: RECord PADding

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

The default value for non-CICS applications is NORECpad.

RECPAD is ignored under CICS.

Syntax

```
       ┌─NORECpad─┐
►►─────┴─RECpad───┴──────────────────────────────────────────────────►◄
```

Usage notes

- The PAD specifier of the OPEN statement can be used to indicate padding for individual files.

REDIR | NOREDIR (C only)

Derivation: REDIRection

REDIR specifies if you can redirect `stdin`, `stdout`, and `stderr` from the command line.

Restriction: This option can only be specified with the `#pragma runopts` directive or the REDIR and NOREDIR compiler options.

The default value for non-CICS applications is REDIR.

REDIR is ignored under CICS.

The default value for AMODE 64 applications is REDIR.

Syntax

```
       ┌─REDIR───┐
►►─────┴─NOREDIR─┴───────────────────────────────────────────────────►◄
```

REDIR
: Specifies that you can redirect `stdin`, `stdout`, and `stderr` from the command line. REDIR applies only if ARGPARSE is also specified or defaulted.

NOREDIR
: Specifies that you cannot redirect `stdin`, `stdout`, and `stderr` from the command line.

For more information

See "ARGPARSE | NOARGPARSE (C only)" on page 15 for a description of ARGPARSE.

RPTOPTS

Derivation: RePorT OPTionS

RPTOPTS

RPTOPTS generates, after an application has run, a report of the runtime options in effect while the application was running. RPTOPTS(ON) lists the declared runtime options in alphabetical order. The report lists the option names and shows where each option obtained its current setting. Language Environment writes options reports only in mixed-case U.S. English.

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

The default value for non-CICS applications is RPTOPTS(OFF).

The default value for CICS applications is RPTOPTS(OFF).

The default value for AMODE 64 applications is RPTOPTS(OFF).

Default	**Value**
Non-CICS	RPTOPTS(OFF)
AMODE 64	RPTOPTS(OFF)

Syntax

```
                  ┌─OFF─┐
►►──RPTOpts──(────┼─────┼──)──────────────────────────────────────────►◄
                  └─ON──┘
```

OFF
Does not generate a report of the runtime options in effect while the application was running.

ON Generates a report of the runtime options in effect while the application was running.

Usage notes

- For AMODE 64 applications, Language Environment writes the options report to `stderr`.
- In some cases, RPTOPTS will 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 runtime option. Under CICS, with RPTOPTS(ON), Language Environment writes the options report to the CESE queue when the transaction ends successfully.
- If the RPTSTG runtime option is specified 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.

Performance consideration

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

For more information

- See “MSGFILE” on page 53 for more information about the MSGFILE runtime option.
- For an example and complete description of the options report, see *z/OS Language Environment Debugging Guide*.

RPTSTG

Derivation: RePorT SToraGe

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

You can use the storage report information to help you set the ANYHEAP, BELOWHEAP, HEAP, HEAP64, HEAPPOOLS, HEAPPOOLS64, IOHEAP64, LIBHEAP64, LIBSTACK, STACK, STACK64, THREADHEAP, THREADSTACK, and THREADSTACK64 runtime options for the best storage tuning. For an example and complete description of the storage report, see *z/OS Language Environment Debugging Guide*.

The default value for non-CICS applications is RPTSTG(OFF).

The default value for CICS applications is RPTSTG(OFF).

The default value for AMODE 64 applications is RPTSTG(OFF).

Syntax

```
                  ┌─OFF─┐
►►──RPTStg──(─────┼─────┼──)──────────────────────────────────────────►◄
                  └─ON──┘
```

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.

CICS consideration

- 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`.
- When a vendor heap manager (VHM) is active, the Language Environment Storage Report will have a text line indicating that the user heap for C/C++ part of the enclave is managed separately. The VHM is expected to write its own storage report to the `stderr` stream.
- In some cases, RPTSTG will not generate the storage report if your application ends abnormally.
- 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 GETMAIN and FREEMAIN requests, respectively.

Performance consideration

- 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*.
- For more information about the MSGFILE runtime option, see “MSGFILE” on page 53.
- 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.

The default value for non-CICS applications is NORTEREUS.

RTEREUS is ignored under CICS.

Syntax

```
     ┌─NORTEREUS─┐
►►───┴─RTEREUS───┴──────────────────────────────────────────────►◄
```

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

RTEREUS
: Initializes the runtime environment to be reusable when the first COBOL program is invoked.

Usage notes

- **Restriction:** Enterprise COBOL programs compiled with the THREAD compiler option do not run with RTEREUS(ON).
- **Guideline:** Avoid using RTEREUS(ON) as a system-level or region-level default. If you do use RTEREUS, use it for specific applications only.
- 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 create multiple enclaves will terminate with error message IGZ0168S. Multiple 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), any attempts to run a C or PL/I main program under Language Environmentt 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 fails.
- If a large number of COBOL programs 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.
- 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.
- IMS consideration—RTEREUS is not recommended for use under IMS.
- 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.

Performance consideration

- You must change STOP RUN statements to GOBACK statements 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 information about specifying this option at the system or region level, *z/OS Language Environment Customization*.
- For information about specifying this option in a CEEUOPT, see *z/OS Language Environment Programming Guide*.
- For more information about IGZERREO, see *z/OS Language Environment Customization*.
- See *z/OS Language Environment Programming Guide* for more information about preinitialization.

SIMVRD (COBOL only)

Derivation: SIMulate Variable length Relative organization Data sets

SIMVRD specifies if your COBOL programs use a VSAM KSDS to simulate variable-length relative organization data sets.

The default value for non-CICS applications is NOSIMVRD.

SIMVRD is ignored under CICS.

Syntax

```
    ┌─NOSIMVRD─┐
►►──┴─SIMVRD───┴──────────────────────────────────────────────────►◄
```

NOSIMVRD
  Do not use a VSAM KSDS to simulate variable-length relative organization.

SIMVRD
  Use a VSAM KSDS to simulate variable-length relative organization.

For more information

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)

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 runtime library routines. The downward-growing stack is allocated only in an XPLINK environment.

The default value for non-CICS applications is STACK(128K,128K,ANYWHERE,KEEP,512K,128K).

The default value for CICS applications is STACK(4K,4080,ANYWHERE,KEEP,4K,4080).

Syntax

```
                                                       ┌─ANYWHERE─┐
►►──STAck──(──┬─────────────┬──,──┬─────────────┬──,──┼──────────┼──,──────────►
              └─usinit_size─┘     └─usincr_size─┘     ├─ANY──────┤
                                                       └─BELOW────┘

   ┌─KEEP─┐
►──┼──────┼──,──┬─────────────┬──,──┬─────────────┬──)─────────────────────────►◄
   └─FREE─┘     └─dsinit_size─┘     └─dsincr_size─┘
```

`usinit_size`
: Determines the initial allocation of the upward-growing stack storage. This value can be specified as *n*, *n*K, or *n*M 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 storage.

  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 storage. This value can be specified as *n*, *n*K, or *n*M 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.

  In the following example:
  - *usincr_size* is specified as 8K
  - The requested size is 9000 bytes
  - The currently allocated stack storage has less than 9000 bytes available

As a result, Language Environment allocates enough storage to hold the 9000 byte request.

If the requested size is smaller than 8K, Language Environment allocates 8K of stack storage.

ANYWHERE|ANY
Specifies that stack storage can be allocated anywhere in storage. If there is no available storage above the line, storage is acquired below the 16 MB line.

BELOW
Specifies that stack storage is allocated below the 16M line in storage.

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

FREE
Specifies that an increment to stack storage is released when the last of the storage within that increment is freed.

dsinit_size
Determines the initial allocation of the downward-growing stack storage. This value can be specified as *n*, *n*K, or *n*M 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 storage. This value can be specified as *n*, *n*K, or *n*M 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.

CICS consideration

- **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).
- The minimum for the initial size is 4K.
- STACK(0), STACK (-0), and STACK (-n) are all interpreted as STACK(4K) under CICS.
- 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 when the storage allocation is below the 16-MB 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 threads 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 runtime option's initial size.
- **Guideline:** The default setting for STACK under z/OS UNIX is STACK=(12K,12K,ANYWHERE,KEEP,512K,128K).

Usage notes

- Applications running with ALL31(OFF) must specify STACK(,,BELOW,,,) to ensure that stack storage is addressable by the application.
- When an application is running in an XPLINK environment, the STACK runtime option is 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 is 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.
- If the initial routine of the Language Environment application is AMODE 24, the STACK runtime option is forced to STACK(,,BELOW). Only the third suboption of the STACK runtime option is changed by this action. No message is issued to indicate this action. However, if a Language Environment runtime options report is generated using the RPTOPTS runtime option, the STACK option is reported as `"Override"` under the `LAST WHERE SET` column.
- The *dsinit_size* and *dsincr_size* values are not the actual amounts of storage obtained. The actual size of the storage obtained is 4K larger (8K if a 4K page alignment cannot be guaranteed) to accommodate the guard page.
- PL/I consideration—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 16 MB. Stack frame extensions are also constrained to 16 MB. Therefore, the size of an automatic aggregate, temporary variable, or dummy argument cannot exceed 16 MB. 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 consideration—The STACK option allocates and manages stack storage for the PL/I main task only.

Performance consideration

You can improve performance with the STACK runtime option by specifying values that minimize the number of times the operating system allocates storage. See “RPTSTG” on page 71 for information about how to generate a report you can use to determine the optimum values for the STACK runtime option.

For more information

- See “ALL31” on page 12 for more information about the ALL31 runtime option.
- See “RPTSTG” on page 71 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, see *z/OS Language Environment Debugging Guide*.
- See “XPLINK” on page 101 for more information about the XPLINK runtime option.

STACK64 (AMODE 64 only)

STACK64 controls the allocation of the thread's stack storage for AMODE 64 applications. Storage required for the common anchor area (CAA) and other control blocks is allocated separately from, and prior to, the allocation of the initial stack segment and the initial heap.

The default value for AMODE 64 applications is STACK64(1M,1M,128M).

Syntax

```
►►──┬─STACK64─┬──(──┬─────────┬──,──┬───────────┬──,──┬─────────┬──)──────────►◄
    └─S64─────┘     └─initial─┘     └─increment─┘     └─maximum─┘
```

initial
: Determines the size of the initial stack segment. The storage is contiguous. This value is specified as *n*M bytes of storage.

increment
: Determines the minimum size of any subsequent increment to the downward-growing stack area. This value is specified as *n*M 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 *n*M bytes of storage. When the maximum size is less than the initial size, *initial* is used as the maximum stack size.

Usage notes

- The 1MB guard page is not included in any of the sizes.
- The maximum stack segment is the maximum of STACK64 initial and maximum sizes.
- The default value of 128MB for the maximum stack size of the STACK64 and THREADSTACK64 runtime options can cause excessive use of system resources (such as real storage) when running a multithreaded application that creates many pthreads. For such applications, it is recommended to use the Language Environment Storage Report (RPTSTG runtime option) to determine your application's actual pthread stack storage usage, and then use the THREADSTACK64 runtime option to set the maximum stack size to a value closer to the actual usage.

Performance consideration

You can improve performance with the STACK64 runtime option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 71 for information about how to generate a report you can use to determine the optimum values for the STACK64 runtime option.

For more information

- See “RPTSTG” on page 71 for more information about the RPTSTG runtime option.
- For more information about heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.
- For more information about using the storage reports generated by the RPTSTG runtime option to tune the stacks, see *z/OS Language Environment Debugging Guide*.

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 runtime 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.

The default value for non-CICS applications is STORAGE(NONE,NONE,NONE,0K).

The default value for CICS applications is STORAGE(NONE,NONE,NONE,0K).

The default value for AMODE 64 applications is STORAGE(NONE,NONE,NONE).

Syntax

```
►►──STOrage──(──┬─────────────────┬──,──┬────────────────┬──,──────────►
                └─heap_alloc_value─┘     └─heap_free_value─┘

►──┬────────────────┬──,──┬──────────────┬──)────────────────────────►◄
   └─dsa_alloc_value─┘     └─reserve_size─┘
```

`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'C2C2C2...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 storage that was acquired dynamically and is composed of a standard register save area and the area available for automatic storage. The dsa_alloc_value has no effect on the XPLINK or AMODE 64 downwards growing stack.

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'C1C1C1...C1'. A *dsa_alloc_value* of 'F' initializes stack storage to X'C6C6C6...C6', 'd' to X'848484...84'.
- Two hex digits without quotations. 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'FEFEFE...FE'. A *dsa_alloc_value* of 00 initializes stack storage to X'00', FF to X'FFFFFF...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*, *n*K, or *n*M bytes of storage. The amount of storage is rounded to the nearest multiple of 8 bytes.

Restriction: This option is ignored in a 64–bit environment.

If you specify *reserve_size* as 0, no reserve segment is allocated. 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 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 determination is made to activate the reserve stack, the reserve size should be set to a minimum of 32K to support Language Environment condition handling and messaging internal routines as well as the user condition handler. When using the reserve stack in a multithreaded environment, it is recommended that the ALL31(ON) and STACK(,,ANY,,,) options also be in effect.

To avoid such an overflow, increase the size of the reserve stack segment with the STORAGE(,,,*reserve_size*) runtime option. The reserve stack segment is not freed until thread termination.

CICS consideration

The out-of-storage condition is not raised under CICS.

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 *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 quote, double it within the string delimited by single quotes or surround it with a pair of double quotation marks. Both of the following are correct ways to specify a single quote:

```
STORAGE('''')
STORAGE("'")
```

  Similarly, double quotation marks must be doubled within a string delimited by double quotation marks, or surrounded by a pair of single quotes. The following are correct ways to specify a double quotation mark:

```
STORAGE("""")
STORAGE('"')
```

- CLEAR is not a valid option for AMODE 64 applications.
- COBOL consideration—If using WSCLEAR in VS COBOL II, STORAGE(00,NONE,NONE,8K) is recommended.

Performance considerations

- Use STORAGE(NONE,NONE,NONE) when you are not debugging.
- 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.

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 the TRACE, UATRACE, DUMP, and UADUMP suboptions of TERMTHDACT.

The TRACE and UATRACE suboptions suppress the dumping of user storage and Language Environment control blocks.

The DUMP and UADUMP suboptions include the dumping of user storage and Language Environment control blocks.

If a message is printed, based upon the TERMTHDACT(MSG) runtime option, the message is for the active condition immediately prior to 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.

The default value for non-CICS applications is TERMTHDACT(TRACE,CESE,96).

The default value for CICS applications is TERMTHDACT(TRACE,CESE,96).

The default value for AMODE 64 applications is TERMTHDACT(TRACE,,96).

Syntax

```
                      ┌─TRACE───┐      ┌─CESE────┐
►►──TERmthdact──(─────┼─────────┼──,───┼─────────┼───,───┬────────────────────┬──)────────►◄
                      ├─QUIET───┤      └─CICSDDS─┘       └─reg_stor_amount────┘
                      ├─MSG─────┤
                      ├─DUMP────┤
                      ├─UATRACE─┤
                      ├─UADUMP──┤
                      ├─UAONLY──┤
                      └─UAIMM───┘
```

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.

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 a U4039 system dump of the user address space. Under CICS, you will get a CICS transaction dump. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space.

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 trace of the active routines on the activation stack, a Language Environment dump, and a U4039 system dump of the user address space. 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 system dump of the user address space. 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 system dump of the user address space. 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.

CESE
Specifies that Language Environment dump output are written to the CESE QUEUE as it has always been. This option is ignored in a 64–bit environment.

CICSDDS
Specifies that Language Environment dump output are written to the new

CICS transaction dump that contains both CICS and CEEDUMP data. This option is ignored in a 64–bit environment.

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 will be rounded up to the nearest multiple of 32. The default amount is 96 bytes.

CICS consideration

All TERMTHDACT output is written to the data queue based on the setting of CESE or CICSDDS. See Table 5 for a summary of the results of the different options that are available.

Table 5. Condition handling of 0Cx ABENDS

Options	**TERMTHDACT(option,CESE,)**	**TERMTHDACT(option,CICSDDS,)**
QUIET	• No output. • ASRA or user ABEND issued.	• No output. • ASRA or user ABEND issued.
MSG	• Message written to CESE queue or MSGFILE. • ASRA or user ABEND issued.	• Message written to CESE queue or MSGFILE. • ASRA or user ABEND issued.
TRACE	• Message written to CESE queue. • Traceback written to CESE queue. • ASRA or user ABEND issued.	• Message written to CESE or MSGFILE. • Traceback included in CICS transaction dump for this ABEND. • ASRA or user ABEND issued.
DUMP	• Message written to CESE queue. • Traceback written to CESE queue. • CEEDUMP to CESE queue. • ASRA or user ABEND issued.	• Invalid suboption combination. Not supported.
UATRACE	• Message written to CESE queue. • Traceback written to CESE queue. • U4039 transaction dump in CICS dump data set. • ASRA or user ABEND issued.	• 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.
UADUMP	• 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.	• Invalid suboption combination. Not supported.
UAONLY	• U4039 transaction dump in CICS dump data set.	• No changes in behavior for CICSDDS.
UAIMM	• U4039 transaction dump in CICS dump data set.	• 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.

Table 6. Handling of software-raised conditions

Options	**TERMTHDACT(option,CESE,)**	**TERMTHDACT(option,CICSDDS,)**
QUIET	• No output. • U4038 abend issued with CANCEL and NODUMP options.	• No output. • U4038 abend issued with CANCEL and NODUMP options.
MSG	• Message written to CESE queue or MSGFILE. • U4038 abend issued.	• Message written to CESE queue or MSGFILE. • U4038 abend issued.
TRACE	• Message written to CESE queue or MSGFILE. • Traceback written to CESE queue. • U4038 abend issued.	• Message written to CESE queue or MSGFILE. • Traceback written to CESE queue. • U4038 abend issued.
DUMP	• Message written to CESE queue or MSGFILE. • Traceback written to CESE queue. • CEEDUMP written to CESE queue. • U4038 abend issued.	• Invalid suboption combination. Not supported.
UATRACE	• Message written to CESE queue or MSGFILE. • Traceback written to CESE queue. • U4039 transaction dump in CICS dump data set. • U4038 abend issued.	• Message written to CESE queue or MSGFILE. • Traceback written to CESE queue. • U4039 transaction dump in CICS dump data set. • U4038 abend issued.
UADUMP	• 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.	• Invalid suboption combination. Not supported.
UAONLY	• U4039 transaction dump in CICS dump data set. • U4038 abend issued.	• No changes in behavior for CICSDDS.
UAIMM	• U4039 transaction dump in CICS dump data set. • U4038 abend issued.	• Incorrect suboption combination. Not supported.

Note:
1. CICS is told about software-raised error conditions for DUMP and TRACE.
2. When assembling a CEEROPT or CEEUOPT, the CICSDDS option cannot be issued with DUMP or UADUMP. This results in a RC=8, CEEXOPT issues an MNOTE, and the setting is forced to TRACE:
3. Language Environment requests a CICS transaction dump via the U4039 abend.
4. See *z/OS Language Environment Runtime Messages* for more complete details regarding the U4039 abend.

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.

If running under a z/OS UNIX shell and Language Environment generates a system dump, a core dump is generated to a file based on the kernel environment variable, _BPXKDUMP.

Usage notes

- A runtime options report is generated and placed at the end of the enclave information whenever the TRACE, UATRACE, DUMP and UADUMP options are invoked.
- 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.
- The environment variable _CEE_DMPTARG allows a sysout class for a dynamically allocated CEEDUMP. You can set the _CEE_DMPTARG value string from a z/OS UNIX shell by:
  - Using the export command
  - Using the C functions setenv() or putenv()
  - Using the ENVAR runtime option

  _CEE_DMPTARG has the following format, `_CEE_DMPTARG=`*`value`*. The *value* is a null-terminated character string, SYSOUT(*x*), that defines a sysout class that Language Environment will set dynamically allocating the CEEDUMP. For example, you can specify: `_CEE_DMPTARG=SYSOUT(A)`

  To set the _CEE_DMPTARG value from a z/OS UNIX shell, you could issue the export command and specify the following run-options, for example:

  ```
  export _CEE_DMPTARG=SYSOUT(A).
  ```

  WHEN _CEE_DMPTARG is not set, then the sysout class will default to SYSOUT(*) for the dynamically allocated CEEDUMP. If the dynamic allocation for the specified SYSOUT class specified by _CEE_DMPTARG should fail, the default, SYSOUT(*) will be used.

For more information

- See “TRACE” on page 94, for more information about the TRACE runtime option.
- For more information about the CEE3DMP service and its parameters, see “CEE3DMP—Generate dump” on page 142.
- See *z/OS Language Environment Programming Guide* for more information about the TERMTHDACT runtime option and condition message.
- For more information about CESE, see *z/OS Language Environment Programming Guide*.

TEST | NOTEST

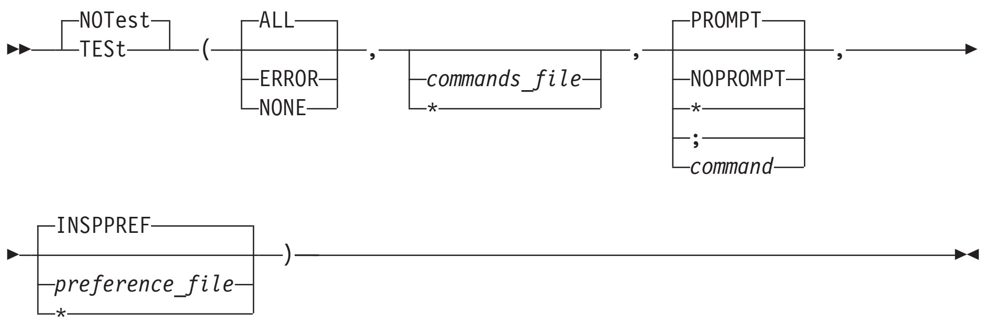
TEST specifies the conditions under which a debug tool (such as the Debug Tool supplied with z/OS) assumes control when the user application is being initialized. Parameters of the TEST and NOTEST runtime options are merged as one set of parameters.

The default value for non-CICS applications is NOTEST(ALL,*,PROMPT,INSPPREF).

The default value for CICS applications is NOTEST(ALL,*,PROMPT,INSPPREF).

The default value for AMODE 64 applications is NOTEST(ALL,*,PROMPT,INSPPREF).

Syntax



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 can 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.

`INSPPREF`
Specifies the default setting for *preference_file*.

`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 a 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.

`* (asterisk—in place of` *`preference_file`*`)`
Specifies that no *preference_file* is supplied.

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 option setting.

Performance consideration

To improve performance, use this option only while debugging.

For more information

See Debug Tool publications for details and examples of the TEST runtime 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.

The default value for non-CICS applications is THREADHEAP(4K,4K,ANY,KEEP).

THREADHEAP is ignored under CICS.

Syntax

```
                                                      ┌─ANYWHERE─┐
►►──THREADHeap──(─┬───────────┬─,─┬───────────┬─,─┼──────────┼─,──────────►
                  └─init_size─┘   └─incr_size─┘   ├─ANY──────┤
                                                  └─BELOW────┘

   ┌─KEEP─┐
►──┼──────┼─)────────────────────────────────────────────────────────────►◄
   └─FREE─┘
```

init_size
The minimum initial size of thread heap storage, and is specified in *n*, *n*K, or *n*M. 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*, *n*K, or *n*M. 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 16M 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.

CICS consideration

Even though this option is ignored under CICS, the default increment size under CICS has changed from 4K (4096 bytes) to 4080 bytes, to accommodate the 16 bytes 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 16M 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 running in 24-bit addressing mode and THREADHEAP(,,ANY,) is in effect, heap storage is allocated below the 16M 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 running in 31-bit addressing mode.
- 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 Fortran for z/OS 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.

The default values for non-CICS applications is THREADSTACK(OFF,4K,4K,ANYWHERE,KEEP,128K,128K).

THREADSTACK is ignored under CICS.

Syntax

```
                     ┌─OFF─┐
►►──THREADSTack──(───┼─────┼──,──┬─────────────┬──,──┬─────────────┬──,──────────►
                     └─ON──┘     └─usinit_size─┘     └─usincr_size─┘

  ┌─ANYWHERE─┐    ┌─KEEP─┐
►─┼──────────┼──,─┼──────┼──,──┬─────────────┬──,──┬─────────────┬──)──────────►◄
  ├─ANY──────┤    └─FREE─┘     └─dsinit_size─┘     └─dsincr_size─┘
  └─BELOW────┘
```

<u>OFF</u>
: Indicates that the allocation suboptions of the STACK runtime option are used for thread stack allocation. Any other suboption specified with THREADSTACK is ignored.

ON
: Controls the stack allocation for each thread, except the initial thread, in a multithreaded 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*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

  The *usinit_size* value 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.

  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*, *n*K, or *n*M 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
Specifies that stack storage can be allocated anywhere in storage. On systems that support bimodal addressing, storage can be allocated either above or below the 16M line. If there is no storage available above the line, Language Environment acquires storage below the line. On systems that do not support bimodal addressing (for example, when VM/ESA is initial program loaded in 370 mode) Language Environment ignores this option and places the stack storage below 16M.

BELOW
Specifies that the stack storage must be allocated below the 16M line in storage that is accessible to 24–bit addressing.

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*, *n*K, or *n*M 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*, *n*K, or *n*M 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.

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 page.
- The downward-growing stack is only initialized in a XPLINK supported environment, that is, batch, TSO, z/OS UNIX, and only when a XPLINK application is active in the enclave. Otherwise the suboptions for the downward-growing stack are ignored.
- 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.

- PL/I MTF consideration—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.
- 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.
- The THREADSTACK option replaces the **NONIPTSTACK** and **NONONIPTSTACK** options.

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.

The default value for AMODE 64 applications is THREADSTACK64(OFF,1M,1M,128M).

Syntax

```
                          ┌─OFF─┐
►►──┬─THREADSTACK64─┬──(──┼─────┼──,─initial─,─increment─,─maximum─)──────►◄
    └─TS64──────────┘     └─ON──┘
```

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 *n*M bytes of storage.

increment
: Determines the minimum size of any subsequent increment to the stack area. This value is specified as *n*M 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 *n*M bytes of storage. When the maximum size is less than the initial size, *initial* is used as the maximum stack size.

Usage notes

- The 1 MB guard page is not included in any of the sizes.
- The maximum thread stack segment is the maximum of THREADSTACK64 initial and maximum sizes.
- The default value of 128 MB for the maximum stack size of the STACK64 and THREADSTACK64 runtime options can cause excessive use of system resources (such as real storage) when running a multithreaded application that creates many pthreads. For such applications, it is recommended to use the Language Environment Storage Report (RPTSTG runtime option) to determine your application's actual pthread stack storage usage, and then use the THREADSTACK64 runtime option to set the maximum stack size to a value closer to the actual usage.

For more information

For more information about heap storage and heap storage tuning with storage report numbers, see *z/OS Language Environment Programming Guide*.

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.

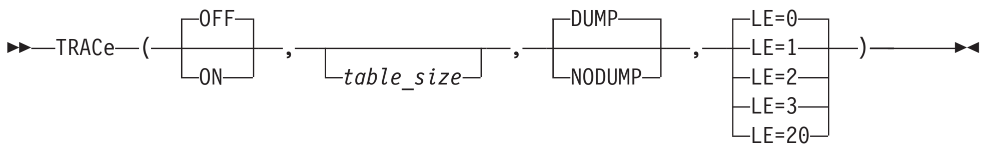
Under normal termination conditions, if TRACE is active and you specify DUMP, only the trace table is written to the dump report, independent of the TERMTHDACT setting. Only one dump is taken for each termination. Under abnormal termination conditions, the type of dump taken (if one is taken) depends on the value of the TERMTHDACT runtime option and if TRACE is active and the DUMP suboption is specified.

The default value for non-CICS applications is TRACE(OFF,4K,DUMP,LE=0).

The default value for CICS applications is TRACE(OFF,4K,DUMP,LE=0).

The default value for AMODE 64 applications is TRACE(OFF,4K,DUMP,LE=0).

Syntax



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 (*n*K or *n*M). The upper limit is 16M - 1 (1666777215 bytes). **Restriction:** This option is ignored in a 64–bit environment 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.

Usage notes

- PL/I MTF consideration—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.
- When running PL/I with POSIX(ON) in effect, 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.
- When TRACE(OFF) is specified, Language Environment activates tracing when a *pthread_create()* is done and a debugger is being used.
- COBOL does not provide any Trace Table Entries.

For more information

- For more information about the dump contents, see "TERMTHDACT" on page 82.

- 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 (see Table 7). CEESGL is unaffected by this option.

TRAP(ON) must be in effect for the ABTERMENC runtime option to have effect. This option is similar to options that were offered by earlier versions of COBOL, C, and PL/I runtime libraries:

- STAE | NOSTAE runtime option of earlier COBOL, C, and PL/I
- SPIE | NOSPIE option offered by earlier C and PL/I

Table 7. TRAP runtime option settings

If ...	Then ...
One 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.
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, or STAE | NOSTAE	TRAP setting takes preference over all others.

The default value for non-CICS applications is TRAP(ON,SPIE).

The default value for CICS applications is TRAP(ON,SPIE).

The default value for AMODE 64 applications is TRAP(ON,SPIE).

Syntax

```
          ┌─ON──┐     ┌─SPIE───┐
►►──TRAP──(─┼─────┼─,─┼────────┼─)──────────────────────────────►◄
          └─OFF─┘     └─NOSPIE─┘
```

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 ignored when specified with the OFF suboption.

NOSPIE
: NOSPIE specifies that Language Environment will not issue the ESPIE macro. When you specify the ON, NOSPIE suboption, Language Environment will handle program interrupts and abends via 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 the Usage notes, IBM highly recommends running TRAP(ON,SPIE) in all environments.

CICS consideration

Since 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

- 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.
- Language Environment requires TRAP(ON) because it uses condition handling internally. As a result, running with TRAP(OFF) for exception diagnosis purposes can cause many side effects. If 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:
  - 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.

  – The COBOL ON SIZE ERROR clause may be disabled for arithmetic statements.
- 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.
- C++ consideration—TRAP(ON) must be in effect in order for the z/OS XL C++ `try`, `throw`, and `catch` condition handling mechanisms to work.
- 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 in a non-CICS environment, 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.

  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.

For more information

- See "ABTERMENC" on page 10 for more information about the ABTERMENC runtime option.
- See "CEESGL—Signal a condition" on page 415 for more information about the CEESGL callable service.
- For more information about the CEEHDLR callable service, see "CEEHDLR—Register user-written condition handler" on page 317.
- 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.

The default for non-CICS applications is UPSI(00000000).

The default for CICS applications is UPSI(00000000).

Syntax

```
►►──UPSI──(──┬──────────┬──)──────────────────────────────────────►◄
             └─nnnnnnnn─┘
```

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).

For more information

- For more information about how COBOL programs access the UPSI switches, 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).

USRHDLR | NOUSRHDLR

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.

The default value for non-CICS applications is NOUSRHDLR.

The default value for CICS applications is NOUSRHDLR().

Syntax

```
     ┌─NOUsrhdlr─┐
►►───┴─USrhdlr───┴──(──┬────────┬──,──┬─────────┬──)────────────────────►◄
                       └─lmname─┘     └─lmname2─┘
```

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.

lmname
: 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.

lmname2
: 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.

CICS consideration

When specifying USRHDLR under CICS, *lmname* and *lmname2* must be defined in the CICS system definition data set (CSD) for your CICS region, rather than using program autoinstall. This includes the sample user-written condition handler CEEWUCHA.

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 run time 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 quotes 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,"")`.
- In SCEESAMP, IBM supplies a sample user-written condition handler, called CEEWUCHA. Under CICS, this handler will give you similar abend codes that existed in certain pre-Language Environment environments. The CEEWUCHA load module needs to be built using CEEWWCHA, which is also provided in SCEESAMP. This handler has support for both COBOL and PL/I; however, it 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 "CEEHDLR—Register user-written condition handler" on page 317.

VCTRSAVE

Derivation: VeCToR environment to be SAVEd

VCTRSAVE specifies if any language in the application uses the vector facility when user-written condition handlers are called.

The default values for non-CICS applications is VCTRSAVE(OFF).

VCTRSAVE is ignored under CICS.

Syntax

```
            ┌─OFF─┐
►►──Vctrsave──(──┼─────┼──)──────────────────────────────────────►◄
            └─ON──┘
```

<u>OFF</u>
: 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.

CICS consideration

VCTRSAVE is ignored under CICS.

z/OS UNIX considerations

The VCTRSAVE option applies to the entire enclave and all threads within the enclave.

Performance consideration

When a condition handler plans to use the vector facility (that is, run any vector instructions), the entire vector environment must 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.

XPLINK

Derivation: e<u>X</u>tra <u>P</u>erformance <u>LINK</u>age

The XPLINK runtime option controls the initialization of the XPLINK environment. XPLINK resources such as the downward-growing stack and the C/C++ run time library are committed only when it is known that an XPLINK program will receive control. If the initial program in the enclave contains at least one function that was compiled using XPLINK conventions, then XPLINK resources will be committed during Language Environment initialization as if XPLINK(ON) had been specified. If the initial program is purely non-XPLINK, then no XPLINK resources will be committed. If it is known that an XPLINK function will be called at some point during the execution of the application, (for example, by calling a DLL function that has been compiled XPLINK), then XPLINK(ON) must be specified so the necessary XPLINK resources are available.

You cannot set this option at the system level, region level, or in the CEEBXITA assembler user exit interface. You can only specify XPLINK on application invocation (command-line interface or _CEE_RUNOPTS environment variable), or as an application default (CEEUOPT, C/C++ #pragma runopts, or PLIXOPT).

The default value for non-CICS applications is XPLINK(OFF).

XPLINK is ignored under CICS.

Syntax

```
              ┌─OFF─┐
►►──XPLink──(─┤     ├─)──────────────────────────────────────────────►◄
              └─ON──┘
```

OFF
: Specifies that no programs containing XPLINK-compiled functions will be run. XPLINK resources will not be committed.

ON
: Specifies that at some time during the execution of an application, an XPLINK-compiled function may be called. XPLINK resources will be committed so that when this occurs, the XPLINK function can properly execute.

Usage notes

The XPLINK(ON) runtime option is provided for application compatibility. It should be specified only for applications that:

- Have a non-XPLINK main(),
- Use XPLINK resources during their execution (for example, calls to an XPLINK function in a DLL), and have been tested to run in an XPLINK environment (that is, they do not use any resources or subsystems that are restricted in an XPLINK environment) Blindly running a non-XPLINK application in an XPLINK environment by specifying the XPLINK(ON) runtime option could result in performance degradation or application abends.

For these reasons, the XPLINK(ON) application should only be specified for individual applications that require it. That is why you cannot set XPLINK(ON) as a system-level or region-level default.

- If the XPLINK runtime option is not specified and the initial program contains at least one XPLINK-compiled part, then the XPLINK runtime option will be forced to (ON). No message will be issued to indicate this action.
- 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. 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.
- 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 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. 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.

Performance considerations

- When XPLINK(ON) is in effect, resources required for the execution of an XPLINK-compiled function are committed. This includes, for example, allocation of a downward-growing stack segment. If no XPLINK functions are invoked, then these are resources that are not available to an XPLINK function.
- If the application contains C or C ++ and XPLINK(ON) is specified, then the XPLINK-compiled version of the C Runtime Library is loaded, which will run on the downward-growing stack. When non-XPLINK functions call C RTL functions in this environment, a swap from the upward-growing stack to the downward-growing stack will occur. This results in additional overhead that could cause performance degradation. Applications that make heavy use of the C RTL from non-XPLINK callers should be aware of this, and if necessary for performance reasons, either run in a pure non-XPLINK environment with XPLINK(OFF), or convert as much of the application to XPLINK as possible and run with XPLINK(ON).
- Applications that consist only of non-XPLINK functions (for example, COBOL or PL/I) should not specify the XPLINK(ON) runtime option, as just turning this on in these applications will result in a performance degradation.

XUFLOW

Derivation: eXponent Under FLOW

XUFLOW specifies if 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.

The default value for non-CICS applications is XUFLOW(AUTO).

The default value for CICS applications is XUFLOW(AUTO).

Syntax

```
              ┌─AUTO─┐
►►──Xuflow──(─┼──────┼─)──────────────────────────────────────────────►◄
              ├─ON───┤
              └─OFF──┘
```

AUTO
An exponent underflow causes or does not cause a program interrupt

dynamically, based on 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.

Usage notes

- PL/I consideration—When setting XUFLOW to OFF, be aware that the semantics of PL/I require the underflow to be signaled.
- z/OS UNIX consideration—The XUFLOW option applies to the entire enclave and all threads within.

Language runtime option mapping

Language Environment provides one set of runtime options for applications. These options are processed at the enclave level and allow you to control many aspects of the Language Environment environment.

Most options are applicable to all Language Environment-conforming languages. In addition, although Language Environment assists migration by mapping current HLL options to Language Environment's options, the runtime options of a particular HLL product might change in the Language Environment-enabled version. However, Language Environment has attempted to maintain runtime options consistently across language products while minimizing required changes within each product.

Tables of pre-Language Environment, HLL-specific options and their Language Environment equivalents are provided in the *z/OS Language Environment Runtime Application Migration Guide*. The mapping is performed automatically by Language Environment, except where noted.

Part 2. Language Environment callable services

The following sections contain detailed information about how to use the Language Environment callable services.

Chapter 3. Quick reference tables for Language Environment services

The following tables are a quick reference of the Language Environment callable services and math services.

Note: Language Environment callable services are not supported for AMODE 64 applications.

Bit manipulation routines

Table 8. Bit manipulation routines

Callable service	Function
CEESICLR	Bit clear. See “CEESICLR—Bit clear” on page 431.
CEESISET	Bit set. See “CEESISET—Bit set” on page 431.
CEESISHF	Bit shift. See “CEESISHF—Bit shift” on page 432.
CEESITST	Bit test. See “CEESITST—Bit test” on page 433.

Condition handling callable services

Table 9. Condition handling callable services

Callable service	Function
CEE3CIB	Returns a pointer to a condition information block (CIB) associated with a given condition token. See “CEE3CIB—Return pointer to condition information block” on page 130.
CEE3GRN	Gets the name of the most current Language Environment-conforming routine in which a condition occurred. See “CEE3GRN—Get name of routine that incurred condition” on page 158.
CEE3GRO	Returns the offset within a failing routine of the most recent condition. See “CEE3GRO—Get offset of condition” on page 163.
CEE3SPM	Queries and modifies the enablement of Language Environment hardware conditions. See “CEE3SPM—Query and modify Language Environment hardware condition enablement” on page 200.
CEE3SRP	Sets the resume point at the next instruction in the calling routine to be used by CEEMRCE. See “CEE3SRP—Set resume point” on page 207.
CEEDCOD	Decodes an existing condition token. See “CEEDCOD—Decompose a condition token” on page 247.
CEEENV	Processes environment variables based on the function codes passed as input. See “CEEENV—Process environmental variables” on page 263.
CEEGQDT	Provides a mechanism by which application code can retrieve the *q_data_token* from the Instance Specific Information (ISI). See “CEEGQDT—Retrieve q_data_token” on page 303.
CEEHDLR	Registers a user condition handler for the current stack frame. See “CEEHDLR—Register user-written condition handler” on page 317.
CEEHDLU	Unregisters a user-written condition handler for the current stack frame. See “CEEHDLU—Unregister user-written condition handler” on page 326.
CEEITOK	Computes the initial condition token for the current condition information block. See “CEEITOK—Return initial condition token” on page 334.

Table 9. Condition handling callable services (continued)

Callable service	Function
CEEMRCE	Resumes execution of a user routine at the location established by CEE3SRP. See "CEEMRCE—Move resume cursor explicit" on page 358.
CEEMRCR	Moves the resume cursor to a position relative to the current position of the handle cursor. See "CEEMRCR—Move resume cursor" on page 363.
CEENCOD	Dynamically constructs a condition token. See "CEENCOD—Construct a condition token" on page 377.
CEESGL	Raises, or signals, a condition to the condition manager. See "CEESGL—Signal a condition" on page 415.

Date and time callable services

Table 10. Date and time callable services

Callable service	Function
CEECBLDY	Converts a string representing a date to a COBOL integer format. See "CEECBLDY—Convert date to COBOL Integer format" on page 212.
CEEDATE	Converts a number representing a Lilian date to a date written in character format. See "CEEDATE—Convert Lilian date to character format" on page 230.
CEEDATM	Converts a number representing the number of seconds since 00:00:00 14 October 1582 to character format. See "CEEDATM—Convert seconds to character timestamp" on page 236.
CEEDAYS	Converts a string representing a date to a Lilian format. See "CEEDAYS—Convert date to Lilian format" on page 241.
CEEDYWK	Calculates the day of the week on which a Lilian date falls. See "CEEDYWK—Calculate day of week from Lilian date" on page 260.
CEEGMT	Computes the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. See "CEEGMT—Get current Greenwich Mean Time" on page 292.
CEEGMTO	Computes values to the calling routine that represent the difference between the local system time and Greenwich Mean Time. See "CEEGMTO—Get offset from Greenwich Mean Time to local time" on page 295.
CEEISEC	Converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582. See "CEEISEC—Convert integers to seconds" on page 330.
CEELOCT	Gets the current local time in three formats. See "CEELOCT—Get current local date or time" on page 346.
CEEQCEN	Queries the century within which Language Environment assumes two-digit year values lie. See "CEEQCEN—Query the century window" on page 382.
CEESCEN	Sets the century in which Language Environment assumes two-digit year values lie. See "CEESCEN—Set the century window" on page 394.
CEESECI	Converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. See "CEESECI—Convert seconds to integers" on page 400.
CEESECS	Converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582. See "CEESECS—Convert timestamp to seconds" on page 405.

Dynamic storage callable services

Table 11. Dynamic storage callable services

Callable service	Function
CEE3RPH	Sets the heading displayed at the top of the storage or options reports that are generated when you specify the RPTSTG(ON) or RPTOPTS(ON) runtime options. See "CEE3RPH—Set report heading" on page 197.
CEECRHP	Creates additional heaps. See "CEECRHP—Create new additional heap" on page 221.
CEECZST	Changes the size of a previously allocated storage element, while preserving its contents. See "CEECZST—Reallocate (change size of) storage" on page 225.
CEEDSHP	Discards an entire heap that you created previously with a call to CEECRHP. See "CEEDSHP—Discard heap" on page 256.
CEEFRST	Frees storage previously allocated by CEEGTST. See "CEEFRST—Free heap storage" on page 282.
CEEGTST	Allocates storage from a heap whose ID you specify. See "CEEGTST—Get heap storage" on page 313.

General callable services

Table 12. General callable services

Callable service	Function
CEE3DLY	Suspends processing of an active enclave for a specified number of seconds. See "CEE3DLY—Suspend processing of the active enclave in seconds" on page 138.
CEE3DMP	Generates a Language Environment formatted dump of the runtime environment and member language libraries. See "CEE3DMP—Generate dump" on page 142.
CEE3INF	Returns to the calling routine information about the system, subsystem, environment, member languages, and version of Language Environment associated with the enclave. See "CEE3INF—Query enclave information" on page 168.
CEE3PRM	Returns to the calling routine the argument string that was specified at program invocation. See "CEE3PRM—Query parameter string" on page 191.
CEE3PR2	Returns to the calling routine the argument string and its associated length that was specified at program invocation. See "CEE3PRM—Query parameter string" on page 191.
CEE3USR	Sets or queries one of two 4-byte fields known as the user area fields. See "CEE3USR—Set or query user area fields" on page 208.
CEEDLYM	Suspends processing of an active enclave for a specified number of milliseconds. See "CEEDLYM—Suspend processing of the active enclave in milliseconds" on page 252.
CEEGPID	Retrieves the version of Language Environment that is currently in use. See "CEEGPID—Retrieve the Language Environment version and platform ID" on page 298.
CEEGTJS	Retrieves the value of an exported JCL symbol. See "CEEGTJS—Retrieves the value of an exported JCL symbol" on page 311.
CEERAN0	Generates a sequence of uniform pseudorandom numbers between 0.0 and 1.0 using the multiplicative congruential method with a user-specified seed. See "CEERAN0—Calculate uniform random numbers" on page 391.
CEETEST	Invokes a tool for debugging. See "CEETEST—Invoke Debug Tool" on page 426.

Initialization and termination services

Table 13. Initialization and termination services

Callable service	Function
CEE3ABD	Stops the enclave with an abend. The abend can be issued either with or without cleanup. See "CEE3ABD—Terminate enclave with an abend" on page 123.
CEE3AB2	Stops the enclave with an abend and a reason code. The abend can be issued either with or without cleanup. See "CEE3AB2—Terminate enclave with an abend and reason code" on page 126.
CEE3GRC	Gets the enclave return code. See "CEE3GRC—Get the enclave return code" on page 150.
CEE3SRC	Sets the enclave return code. See "CEE3SRC—Set the enclave return code" on page 206.

Locale callable services

Table 14. Locale callable services

Callable service	Function
CEEFMON	Formats monetary strings. See "CEEFMON—Format monetary string" on page 274.
CEEFTDS	Formats time and date into a character string. See "CEEFTDS—Format time and date into character string" on page 286.
CEELCNV	Queries locale numeric conventions. See "CEELCNV—Query locale numeric conventions" on page 340.
CEEQDTC	Queries locale date and time conventions and returns the specified format information. See "CEEQDTC—Query locale date and time conventions" on page 385.
CEEQRYL	Queries the active locale environment. See "CEEQRYL—Query active locale environment" on page 389.
CEESCOL	Compares the collation weights of two strings. See "CEESCOL—Compare collation weight of two strings" on page 397.
CEESETL	Sets the locale operating environment. "CEESETL—Set locale operating environment" on page 411.
CEESTXF	Transforms string characters into collation weights. See "CEESTXF—Transform string characters into collation weights" on page 420.

Math services

Table 15. Math services

Math service	Function
CEESxABS	Absolute value. See "CEESxABS—Absolute value" on page 437.
CEESxACS	Arccosine. See "CEESxACS—Arccosine" on page 439.
CEESxASN	Arcsine. See "CEESxASN—Arcsine" on page 440.
CEESxATH	Hyperbolic arctangent. See "CEESxATH—Hyperbolic arctangent" on page 441.
CEESxATN	Arctangent. See "CEESxATN—Arctangent" on page 442.
CEESxAT2	Arctangent of two arguments. See "CEESxAT2—Arctangent2" on page 443.
CEESxCJG	Conjugate complex. See "CEESxCJG—Conjugate of complex" on page 444.
CEESxCOS	Cosine. See "CEESxCOS—Cosine" on page 445.
CEESxCSH	Hyperbolic cosine. See "CEESxCSH—Hyperbolic cosine" on page 446.
CEESxCTN	Cotangent. See "CEESxCTN—Cotangent" on page 448.

Table 15. Math services (continued)

Math service	**Function**
CEESxDIM	Positive difference. See “CEESxDIM—Positive difference” on page 449.
CEESxDVD	Division. See “CEESxDVD—Floating-point complex divide” on page 450.
CEESxERC	Error function complement. See “CEESxERC—Error function complement” on page 451.
CEESxERF	Error function. See “CEESxERF—Error function” on page 452.
CEESxEXP	Exponential (base e). See “CEESxEXP—Exponential base e” on page 453.
CEESxGMA	Gamma function. See “CEESxGMA—Gamma function” on page 454.
CEESxIMG	Imaginary part of complex. See “CEESxIMG—Imaginary part of complex” on page 455.
CEESxINT	Truncation. See “CEESxINT—Truncation” on page 456.
CEESxLGM	Log gamma function. See “CEESxLGM—Log gamma” on page 457.
CEESxLG1	Logarithm base 10. See “CEESxLG1—Logarithm base 10” on page 458.
CEESxLG2	Logarithm base 2. See “CEESxLG2—Logarithm base 2” on page 459.
CEESxLOG	Logarithm base e. See “CEESxLOG—Logarithm base e” on page 460.
CEESxMLT	Floating-point complex multiplication. See “CEESxMLT—Floating-point complex multiply” on page 461.
CEESxMOD	Modular arithmetic. See “CEESxMOD—Modular arithmetic” on page 462.
CEESxNIN	Nearest integer. See “CEESxNIN—Nearest integer” on page 463.
CEESxNWN	Nearest whole number. See “CEESxNWN—Nearest whole number” on page 464.
CEESxSGN	Transfer of sign. See “CEESxSGN—Transfer of sign” on page 465.
CEESxSIN	Sine. See “CEESxSIN—Sine” on page 466.
CEESxSNH	Hyperbolic sine. See “CEESxSNH—Hyperbolic sine” on page 468.
CEESxSQT	Square root. See “CEESxSQT—Square root” on page 469.
CEESxTAN	Tangent. See “CEESxTAN—Tangent” on page 470.
CEESxTNH	Hyperbolic tangent. See “CEESxTNH—Hyperbolic tangent” on page 472.
CEESxXPx	Exponential (**). See “CEESxXPx—Exponentiation” on page 473.

Message handling callable services

Table 16. Message handling callable services

Callable service	**Function**
CEECMI	Stores message insert data. See “CEECMI—Store and load message insert data” on page 216.
CEEMGET	Retrieves a message corresponding to a condition token returned from a callable service. See “CEEMGET—Get a message” on page 350.
CEEMOUT	Writes a specified message to the message string file. See “CEEMOUT—Dispatch a message” on page 355.
CEEMSG	Retrieves a message corresponding to a condition token received and writes it to the message file. See “CEEMSG—Get, format, and dispatch a message” on page 373.

National Language Support callable services

Table 17. National Language Support and National Language Architecture callable services

Callable service	Function
CEE3CTY	Changes or queries the current national country setting. See “CEE3CTY—Set default country” on page 133.
CEE3LNG	Changes or queries the current national language. See “CEE3LNG—Set national language” on page 172.
CEE3MCS	Gets the default currency symbol for a country specified in *country_code*. See “CEE3MCS—Get default currency symbol” on page 179.
CEE3MC2	Gets the default currency symbol and the international currency symbol for a country specified in *country_code*. See “CEE3MC2—Get default and international currency symbols” on page 182.
CEE3MDS	Gets the default decimal separator for the country specified in *country_code*. See “CEE3MDS—Get default decimal separator” on page 185.
CEE3MTS	Gets the default thousands separator for the country that you specify in *country_code*. See “CEE3MTS—Get default thousands separator” on page 188.
CEEFMDA	Gets the default date picture string for a country specified in *country_code*. See “CEEFMDA—Get default date format” on page 268.
CEEFMDT	Gets the default date and time picture strings for the country specified in *country_code*. See “CEEFMDT—Get default date and time format” on page 271.
CEEFMTM	Gets the default time picture string for the country specified in *country_code*. See “CEEFMTM—Get default time format” on page 279.

Chapter 4. Using Language Environment callable services

Restriction: Language Environment services are not supported for AMODE 64 applications.

This topic provides syntax and examples of Language Environment callable services, which you can invoke from applications generated with the following IBM compiler products:
- IBM z/OS XL C/C++
- C/C++ for MVS/ESA
- IBM SAA AD/Cycle C/370
- Enterprise COBOL for z/OS
- Enterprise PL/I for z/OS
- IBM COBOL for OS/390 & VM
- IBM COBOL for MVS & VM
- IBM PL/I for MVS & VM

You can invoke Language Environment callable services from assembler programs using the CEEENTRY and associated macros. While you cannot call these services directly from Fortran programs, you can use the Fortran routines AFHCEEN and AFHCEEF to invoke most of these services. (See *Fortran Run-time Migration Guide.)* You can use the other languages to perform these services on behalf of a Fortran program.

Note:
1. Customers using this information may not have pre-Language Environment runtime libraries.
2. You can use DYNAMIC calls from VS COBOL II programs to Language Environment Date/Time callable services. You can not use DYNAMIC calls from VS COBOL II programs to other Language Environment callable services. You can not use STATIC calls from VS COBOL II programs to any Language Environment callable services.
3. A Language Environment callable service used by application programmers can also be called an application writer interface (AWI).

Language Environment callable services provide functions that pre-Language Environment runtime libraries might not provide. You can use these services alone or with Language Environment runtime options, which customize your runtime environment. For guidelines about writing your own callable services, see *z/OS Language Environment Programming Guide*.

Locating callable service information

The following sections help you use the Language Environment callable services. For a summary of Language Environment callable services, see Chapter 1, "Summary of Language Environment runtime options," on page 3.
- "General usage notes for callable services" on page 114
- "Invoking callable services" on page 114
- "Data type definitions" on page 119
- Chapter 5, "Callable services," on page 123

General usage notes for callable services

- You can invoke callable services from any Language Environment-conforming HLL except where otherwise noted.
- You might receive feedback codes from services other than the one you are invoking. This is because the callable services invoke other callable services that might return a feedback code.
- Callable services that are intended to be available on any platform that Language Environment supports are prefixed with CEE. Callable services defined only for S/370 are prefixed with CEE3.
- Routines that invoke callable services do not need to be in 31-bit addressing mode. 31-bit addressing mode switching is performed implicitly without any action required by the calling routine, if you specify the ALL31(OFF) runtime option (see "ALL31" on page 12).

  However, if AMODE switching occurs and your program makes many calls to Language Environment callable services, the switching time can slow down your application. Run with AMODE(31), if possible, to avoid unnecessary mode switching.
- Under Language Environment, all parms are passed by reference, indirectly. The code in Figure 2 and Figure 3 might cause unpredictable results:

```
CALL CEEDATE USING X,Y,Z.  **Invalid**
```

Figure 2. An invalid COBOL CALL that omits the fc parameter

```
DCL CEEDATE ENTRY OPTIONS(ASM);
CALL CEEDATE(x,y,z);      /* invalid */
```

Figure 3. An invalid PL/I CALL that omits the fc parameter

Figure 4 and Figure 5 illustrate valid calls. The COBOL examples are valid only for COBOL for MVS & VM Release 2 or later.

```
CALL CEEDATE USING X,Y,Z,OMITTED.
CALL CEEDATE USING X,Y,Z,FC.
```

Figure 4. Valid COBOL CALLs that use the optional fc parameter

```
DCL CEEDATE ENTRY(*,*,*,* OPTIONAL) OPTIONS(ASM);
CALL CEEDATE(x,y,z,*);    /* valid   */
CALL CEEDATE(x,y,z,fc);   /* valid   */
```

Figure 5. Valid PL/I CALLs that use the optional fc parameter

Invoking callable services

You can invoke Language Environment callable services from assembler routines, HLL-generated object code, HLL library routines, other Language Environment library routines, and user-written HLL calls. When you want to access Language Environment library routines, you can use the same type of user-written HLL calls and functions you currently use for your HLL applications.

This topic provides syntax and examples to help you request callable services from C/C++, COBOL, and PL/I.

Header, copy, or include files

Many of the programming examples in this section imbed header, copy, or include files. (Whether you call the files header files, copy files, or include files depends on the language you are using.) These files can save you time by providing declarations for symbolic feedback codes and Language Environment callable services that you would otherwise need to code in your program. They can also help you reduce errors by verifying correct usage of Language Environment callable services at compile time.

Table 18 summarizes the names and descriptions of the files imbedded in the callable service examples in this section.

Table 18. Files used in C/C++, COBOL, and PL/I examples

File name	Description
ceeedcct.h	C declarations for Language Environment symbolic feedback codes
leawi.h	Declarations of Language Environment callable services and OMIT_FC, which is used to explicitly omit the *fc* parm, for routines written in C/C++
CEEIGZCI	Declarations for condition information block
CEEIGZCT	Declarations for Language Environment symbolic feedback codes
CEEIGZLC	Locale category constants LC-ALL, LC-COLLATE, LC-CTYPE, LC-MESSAGES, LC-MONETARY, LC-NUMERIC, LC-TIME, and version_info structure.
CEEIGZTD	TD-STRUCT structure needed for CEEFTDS calls
CEEIGZNM	NM-STRUCT structure needed for CEELCNV calls
CEEIGZN2	NM-STRUCT structure needed for ISO/IEC 9899:1999 (C99) support of CEELCNV calls
CEEIGZDT	DTCONV structure needed for CEEQDTC calls
CEEIGZD2	DTCONV structure needed for C99 support of CEEQDTC calls
CEEIBMAW	Language Environment callable service declarations for routines written in PL/I
CEEIBMCI	Declarations for condition information block
CEEIBMCT	Declarations for Language Environment symbolic feedback codes
CEEIBMLC	Locale category constants LC-ALL, LC-COLLATE, LC-CTYPE, LC-MESSAGES, LC-MONETARY, LC-NUMERIC, LC-TIME, and version_info structure.
CEEIBMTD	TD-STRUCT structure needed for CEEFTDS calls
CEEIBMNM	NM-STRUCT structure needed for CEELCNV calls
CEEIBMN2	NM-STRUCT structure needed for C99 support of CEELCNV calls
CEEIBMDT	DTCONV structure needed for CEEQDTC calls
CEEIBMD2	DTCONV structure needed for C99 support of CEEQDTC calls
CEEFORCT	Fortran declarations for Language Environment symbolic feedback codes

Note:

1. A symbolic feedback code is a symbolic representation of a condition token.

2. PL/I routines that imbed CEEIBMAW require the MACRO compiler option.
3. COBOL does not require declarations of external programs; therefore, you do not need declarations of Language Environment callable services for programs written in COBOL.
4. CEEIBMxx parts support PL/I applications.
5. CEEIGZxx parts support COBOL applications.

On z/OS, these files are contained in the Language Environment SCEESAMP partitioned data set, except for `leawi.h`, which is contained in the SCEEH.H partitioned data set. You can imbed these files using the statement appropriate for the language your routine is written in, as shown in Table 19. Examples of these statements are shown in the following topics on syntax.

Table 19. Imbedding files in your routines

To imbed a file in a...	Use statement...
C/C++ routine	`#include`
COBOL program	`COPY`
PL/I routine	`%include`

Sample programs

Sample C/C++, COBOL, and PL/I programs are provided for most of the callable services. Sample Fortran programs are not provided because you cannot invoke Language Environment callable services from Fortran programs. However, you can use other languages to perform these services on behalf of a Fortran program. These sample programs are also provided online with the Language Environment product for your convenience. On MVS, the sample routines are located in the SCEESAMP partitioned data set.

C/C++ syntax

In C or C++, use the following syntax to invoke a Language Environment callable service with a feedback code in effect:

Syntax

```
►►──ceexxxx──(parm1, parm2, ...parmn, fc);────────────────────────────►◄
```

Note: "..." is "and so on," not the C ellipsis operator.

Use the following syntax to invoke callable services with an omitted feedback code parameter:

Syntax

```
►►──CEExxxx──(parm1, parm2, ...parmn, NULL);──────────────────────────►◄
```

See “Parameter list for invoking callable services” on page 119 for a description of this syntax.

Language Environment callable services always have a return type of *void* and should be prototyped as such.

Input strings for callable services are **not** NULL terminated in C/C++.

You can use the SCEEH.H file `leawi.h` to declare Language Environment callable services, in conjunction with a C/C++ call to a Language Environment callable service, as shown in Figure 6.

```
#include <leawi.h>
int main(void)
{
 CEExxxx(parm1, parm2, ... parmn, fc);
 }
```

Figure 6. Sample callable services invocation syntax for C/C++

To help in checking the success of your applications, Language Environment provides FBCHECK in the `ceeedcct.h` file. FBCHECK compares a feedback code against a condition token you supply to determine whether you receive the feedback code you should.

COBOL syntax

In COBOL, use the following syntax to invoke Language Environment callable services:

Syntax

```
►►──CALL "CEExxxx" USING── parm1, parm2, ...parmn, fc──────────────────────►◄
```

You can call Language Environment services either statically or dynamically from COBOL applications.

The Language Environment callable services are architected as low maintenance stubs that branch to the actual runtime routine that performs the service. Static CALL to these stubs is the preferred choice for best performance. This avoids the overhead of a COBOL dynamic call without the usual maintainability disadvantage of a more robust statically link-edited routine.

See “Parameter list for invoking callable services” on page 119 for a description of this syntax.

You can use the SCEESAMP file CEEIGZCT to declare symbolic Language Environment feedback codes, in conjunction with a COBOL call to a Language Environment callable service to return a feedback code, as shown in Figure 7 on page 118.

```
COPY CEEIGZCT.
CALL "CEExxxx" USING parm1, parm2, ... parmn, fc
```

Figure 7. Sample callable services invocation syntax for COBOL

COBOL for MVS & VM Release 2 or later also allows you to omit arguments. In place of the argument, use the OMITTED parameter, as shown below. See Figure 2 on page 114 and Figure 4 on page 114 for an example of omitting parameters in COBOL.

Syntax

```
►►──CALL "CEExxxx" USING── parm1, parm2, ...parmn, OMITTED──────────────────►◄
```

PL/I syntax

In PL/I, use the following syntax to invoke a Language Environment callable service with a feedback code in effect:

Syntax

```
►►──CALL CEExxxx──(parm1, parm2, ...parmn, *);──────────────────────────────►◄
```

PL/I also allows you to omit arguments. In place of the argument, code an asterisk (*), as shown below:

Syntax

```
►►──CALL CEExxxx──(*, *, ...*, );───────────────────────────────────────────►◄
```

Note that you cannot invoke callable services as function references.

See “Parameter list for invoking callable services” on page 119 for a description of this syntax.

The value of register 15 upon return from any Language Environment callable service is undefined. Use of OPTIONS(RETCODE) is not recommended, because a subsequent use of the PLIRETV built-in function returns an undefined value.

If you code your own declarations for the Language Environment callable services, be sure to specify all required arguments in the CALL statement. Figure 3 on page 114 illustrates a call in which the feedback code to CEEDATE was incorrectly omitted.

You can use the SCEESAMP file CEEIBMAW to declare Language Environment callable services with a PL/I call to a Language Environment callable service, as Figure 8 shows.

```
%INCLUDE CEEIBMAW;
CALL CEExxxx(parm1, parm2, ... parmn, fc);
```

Figure 8. Sample callable services invocation syntax for PL/I

Parameter list for invoking callable services

This section describes the syntax and parameters you need to invoke Language Environment callable services.

CEExxxx
The name of the callable service. By including a reference to a header file in your code, you can avoid declaring each callable service as an external entry. See "Header, copy, or include files" on page 115 for these file names.

*parm1 parm2 ... parm***n**
Optional or required parameters passed to or returned from the called service. Some callable service parameters are optional in C/C++ and PL/I only. If you do not want to pass the parm or you do not want the return value, you can omit the parm, using the appropriate syntax to indicate that the parm is omitted.

fc A feedback code that indicates the result of the service. *fc* can be omitted when you use C/C++, PL/I, and COBOL (COBOL for MVS & VM Release 2 or later).

If you specify *fc* as an argument, feedback information in the form of a condition token is returned to the calling routine. The condition token indicates whether the service completed successfully or whether a condition was encountered while the service was running. In Language Environment you can decode the condition token so that it can be acted on.

If you omit *fc* as an argument, using the appropriate syntax to indicate *fc* is omitted, the condition is signaled if the service was not successful.

Because callable services call other services, these other services might generate feedback codes.

Data type definitions

Parameters in Language Environment are defined as specific data types, such as:

- Fullword binary integer
- Short floating-point hexadecimal
- Long floating-point hexadecimal
- Fixed-length character string with a predefined length
- Entry variable
- Character string with a halfword prefix indicating its current length

Table 20 on page 120, Table 21 on page 121, and Table 22 on page 122 contain data type definitions and their descriptions for C/C++, COBOL, and PL/I, respectively.

C/C++ data type definitions

Table 20 on page 120 includes data type definitions and their descriptions for C/C++.

Table 20. Data type definitions for C/C++

Data type	Description	C/C++
INT2	A 2-byte signed integer	signed short
INT4	A 4-byte signed integer	signed int
FLOAT4	A 4-byte single-precision floating-point number	float
FLOAT8	An 8-byte double-precision floating-point number	double
FLOAT16	A 16-byte extended-precision floating-point number	long double
COMPLEX8	Short floating-point complex hex number: an 8-byte complex number, whose real and imaginary parts are each 4-byte single-precision floating-point numbers	Not available
COMPLEX16	Long floating-point complex hex number: a 16-byte complex number, whose real and imaginary parts are each 8-byte double-precision floating-point numbers	Not available
COMPLEX32	Extended floating-point hex number: a 32-byte complex number, whose real and imaginary parts are each 16-byte extended-precision floating-point numbers	Not available
POINTER	A platform-dependent address pointer	void *
CHAR*n*	A string (character array) of length *n*	char[n]

VSTRING	A halfword length-prefixed character string (for input); fixed-length 80-character string (for output)	`struct _VSTRING(`
		` _INT2 short length;`
		` char  string[1];`
		` }string-in;`
		`char string_out[80];`

FEED_BACK	A mapping of the condition token (fc)	`Case 1:`
		`typedef struct {`
		`   short  tok_sev     ;`
		`   short  tok_msgno   ;`
		`   int    tok_case :2,`
		`          tok_sever:3,`
		`          tok_ctrl :3 ;`
		`   char   tok_facid[3];`
		`   int    tok_isi     ;`
		`        }   _FEEDBACK ;`
		
		`Case 2:`
		`typedef struct {`
		`   short  tok_sev     ;`
		`   short  tok_msgno   ;`
		`   int    tok_class_code :2,`
		`          tok_cause_code:3,`
		`          tok_ctrl :3 ;`
		`   char   tok_facid[3];`
		`   int    tok_isi     ;`
		`        }   _FEEDBACK ;`

CEE_ENTRY	An HLL-dependent entry constant	FUNCTION POINTER

COBOL data type definitions

Table 21 includes data type definitions and their descriptions for COBOL.

Table 21. Data type definitions for COBOL

Data type	Description	COBOL
INT2	A 2-byte signed integer	PIC S9(4) USAGE IS BINARY
INT4	A 4-byte signed integer	PIC S9(9) USAGE IS BINARY
FLOAT4	A 4-byte single-precision floating-point number	COMP-1
FLOAT8	An 8-byte double-precision floating-point number	COMP-2
FLOAT16	A 16-byte extended-precision floating-point number	Not available
COMPLEX8	Short floating-point complex hex number: an 8-byte complex number, whose real and imaginary parts are each 4-byte single-precision floating-point numbers	Not available
COMPLEX16	Long floating-point complex hex number: a 16-byte complex number, whose real and imaginary parts are each 8-byte double-precision floating-point numbers	Not available
COMPLEX32	Extended floating-point hex number: a 32-byte complex number, whose real and imaginary parts are each 16-byte extended-precision floating-point numbers	Not available
POINTER	A platform-dependent address pointer	USAGE IS POINTER
CHAR*n*	A string (character array) of length *n*	PIC X(n)

VSTRING	A halfword length-prefixed character string (for input); fixed-length 80-character string (for output)	`01 STRING-IN.`
		`  02 LEN      PIC S9(4) USAGE IS BINARY.`
		`  02 TXT      PIC X(N).`
		`01 STRING-OUT PIC X(80).`

FEED_BACK	A mapping of the condition token (fc)	`Case 1:`
		`01 FC`
		`  02 SEV        PIC S9(4) USAGE IS BINARY.`
		`  02 MSGNO      PIC S9(4) USAGE IS BINARY.`
		`  02 FLGS       PIC X(1).`
		`  02 FACID      PIC X(3).`
		`  02 ISI        PIC X(4).`
		`Case 2:`
		`01 FC`
		`  02 CLASS-CODE PIC 9(4) USAGE IS BINARY.`
		`  02 CAUSE-CODE PIC 9(4) USAGE IS BINARY.`
		`  02 FLGS       PIC X(1).`
		`  02 FACID      PIC X(3).`
		`  02 ISI        PIC X(4).`

CEE_ENTRY	An HLL-dependent entry constant	USAGE IS PROCEDURE-POINTER

PL/I data type definitions

Table 22 includes data type definitions and their descriptions for PL/I.

Table 22. Data type definitions for PL/I

Data type	Description	PL/I
INT2	A 2-byte signed integer	REAL FIXED BINARY (15,0)
INT4	A 4-byte signed integer	REAL FIXED BINARY (31,0)
FLOAT4	A 4-byte single-precision floating-point number	REAL FLOAT BINARY (21) or REAL FLOAT DECIMAL (6)
FLOAT8	An 8-byte double-precision floating-point number	REAL FLOAT BINARY (53) or REAL FLOAT DECIMAL (16)
FLOAT16	A 16-byte extended-precision floating-point number	REAL FLOAT DECIMAL (33) or REAL FLOAT BINARY (109)
COMPLEX8	Short floating-point complex hex number: an 8-byte complex number, whose real and imaginary parts are each 4-byte single-precision floating-point numbers	COMPLEX FLOAT DECIMAL (6)
COMPLEX16	Long floating-point complex hex number: a 16-byte complex number, whose real and imaginary parts are each 8-byte double-precision floating-point numbers	COMPLEX FLOAT DECIMAL (16)
COMPLEX32	Extended floating-point hex number: a 32-byte complex number, whose real and imaginary parts are each 16-byte extended-precision floating-point numbers	COMPLEX FLOAT DECIMAL (33)
POINTER	A platform-dependent address pointer	POINTER
CHAR*n*	A string (character array) of length *n*	CHAR(n)

VSTRING	A halfword length-prefixed character string (for input); fixed-length 80-character string (for output)	`DCL string_in  CHAR(n) VARYING;`
		`DCL string_out CHAR(80);`

FEED_BACK	A mapping of the condition token (fc)	`Case 1:`
		`DCL 1 FEEDBACK BASED,`
		`      3 SEVERITY   FIXED BINARY (15),`
		`      3 MSGNO      FIXED BINARY (15),`
		`      3 FLAGS,`
		`        5 CASE     BIT  (2),`
		`        5 SEVERITY BIT  (3),`
		`        5 CONTROL  BIT  (3),`
		`        3 FACID    CHAR (3),`
		`        3 ISI      FIXED BINARY (31);`
		
		`Case 2:`
		`DCL 1 FEEDBACK BASED,`
		`      3 CLASS_CODE FIXED BINARY (15),`
		`      3 CAUSE_CODE FIXED BINARY (15),`
		`      3 FLAGS,`
		`        5 CASE     BIT (2),`
		`        5 SEVERITY BIT (3),`
		`        5 CONTROL  BIT (3),`
		`        3 FACID    CHAR (3),`
		`        3 ISI      FIXED BINARY (31);`

CEE_ENTRY	An HLL-dependent entry constant	ENTRY

Chapter 5. Callable services

This section lists the Language Environment callable services and shows examples of how to use them in C/C++, COBOL, and PL/I.

CEE3ABD—Terminate enclave with an abend

CEE3ABD requests that Language Environment terminate the enclave with an abend. The issuing of the abend can be either with or without clean-up. There is no return from this service, nor is there any condition associated with it.

Syntax

```
►►—CEE3ABD—(—abcode—,—clean-up—)——————————————————————————►◄
```

abcode **(input)**
A fullword integer, no greater than 4095, specifying the abend code that is issued. Under CICS, this fullword integer is converted to EBCDIC.

clean-up **(input)**
Indicates if the abend should result in clean-up of the enclave's resources. The acceptable values for *clean-up* are as follows:

0	Issue the abend without clean-up
1	Issue the abend with normal enclave termination processing
2	Issue the abend with enclave termination processing honoring the TERMTHDACT runtime option for taking a system dump of the user address space but suppressing the CEEDUMP
3	Issue the abend with enclave termination processing suppressing both the CEEDUMP and system dump if requested by the TERMTHDACT runtime option
4	Issue the abend with enclave termination processing honoring the TERMTHDACT runtime option for taking a CEEDUMP but suppressing the system dump
5	Issue the abend with enclave termination processing forcing a system dump of the user address space and suppressing the CEEDUMP

If an illegal value for *clean-up* is passed, the abend is issued without clean-up.

If *clean-up* is 0, no Language Environment dump is generated. A system dump, however, is requested when issuing the abend. Under CICS, a transaction dump is taken. To get a dump under CMS, specify FILEDEF SYSABEND PRINTER or FILEDEF SYSUDUMP PRINTER.

If *clean-up* is 0, Language Environment condition handling is disabled for the current enclave and termination activities are not performed. Event handlers are not driven; Debug Tool is not invoked; user exits are not invoked; and user-written condition handlers are not invoked.

When *clean-up* is 1, the abend is processed in the same manner as if it were a non-Language Environment abend. Its processing is affected by the ABPERC and TRAP options, the filedef abends percolated in the assembler user exit, and other elements of the environment related to abend processing. In particular, the condition handler can intercept the abend and give the application a chance to handle the abend. If the condition remains unhandled, normal termination activities are performed: information such as a Language Environment dump is produced, depending on the setting of the TERMTHDACT option; event handlers are driven; Debug Tool is invoked; and user exits are invoked. Assembler user exit settings control if the application actually terminates with an abend.

When *clean-up* is 2, it follows the abend processing as when *clean-up* is 1 except the CEEDUMP is suppressed.

When *clean-up* is 3, it follows the abend processing as when *clean-up* is 1, however CEEDUMP and system dumps are both suppressed

When *clean-up* is 4, it follows the abend processing as when *clean-up* is 1, however only the system dumps are suppressed.

When *clean-up* is 5, it follows the abend processing as when *clean-up* is 1, however forcing a system dump to be taken and suppressing the CEEDUMP.

z/OS UNIX considerations

- In a multithreaded environment, CEE3ABD applies to the enclave.
- When ALL31(ON) is in effect, Language Environment allocates thread-specific control blocks from the anywhere heap.

Usage notes

1. **Recommendation:** Language Environment abend codes are usually in the range of 4000 (X'FA0') to 4095 (X'FFF'). You should use the range of 0 to 3999 to avoid confusion with Language Environment abend codes. The value specified for the abend code is passed directly to the ABEND macro without further verification.
2. When TRAP(OFF) is specified, CEE3ABD behaves in a similar manner to `clean-up 0`.
3. In a non-CICS environment, a system dump of the user address space is taken only if a SYSMDUMP, SYSUDUMP, or SYSABEND DD card is present.
4. Users can look at CEE3AB2 to include user reason codes separately from abend codes.

For more information

- For more information about the ABPERC runtime option, see "ABPERC" on page 9.
- For more information about the TRAP runtime option, see "TRAP" on page 96.
- For more information about the TERMTHDACT runtime option, see "TERMTHDACT" on page 82.
- For more information about the CEE3AB2 callable service, see "CEE3AB2—Terminate enclave with an abend and reason code" on page 126.

Examples

1. An example of CEE3ABD called by C/C++.

```
/*Module/File Name: EDC3ABD   */
 /****************************************************/
 /*                                                  */
 /*  Licensed Materials - Property of IBM            */
 /*                                                 */
 /*  5688-198 (C) Copyright IBM Corp. 1991, 1995     */
 /*  All rights reserved                             */
 /*                                                 */
 /*  US Government Users Restricted Rights - Use,    */
 /*  duplication or disclosure restricted by GSA     */
 /*  ADP Schedule Contract with IBM Corp.            */
 /*                                                 */
 /***************************************************/

#include <leawi.h>

int main(void) {

  _INT4 code, timing;

  code = 1234; /* Abend code to issue */
  timing = 0;

  CEE3ABD(&code,&timing);
}
```

2. An example of CEE3ABD called by COBOL.

```
CBL LIB,QUOTE
      *************************************************
            *Module/File Name: IGZT3ABD
      **********************************************
      ** CBL3ABD - Call CEE3ABD to terminate the  **
      **           enclave with an abend          **
      **********************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3ABD.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ABDCODE                 PIC S9(9) BINARY.
       01  TIMING                  PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLMGET.

      **********************************************
      ** 3415 is the abend code to be issued,     **
      ** a timing of zero requests an abend       **
      ** without clean-up                         **
      **********************************************
                MOVE 3415 TO ABDCODE.
                MOVE 0 TO TIMING.
                CALL "CEE3ABD" USING ABDCODE , TIMING.

                GOBACK.
CBL LIB,QUOTE
      *************************************************
      *Module/File Name: IGZT3ABD
      **********************************************
      ** CBL3ABD - Call CEE3ABD to terminate the  **
      **           enclave with an abend          **
      **********************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3ABD.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ABDCODE                 PIC S9(9) BINARY.
       01  TIMING                  PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLMGET.
```

```
          ***********************************************
          ** 3415 is the abend code to be issued,     **
          ** a timing of zero requests an abend       **
          ** without clean-up                         **
          ***********************************************
                    MOVE 3415 TO ABDCODE.
                    MOVE 0 TO TIMING.
                    CALL "CEE3ABD" USING ABDCODE , TIMING.

                    GOBACK.
```

3. An example of CEE3ABD called by PL/I.

```
*PROCESS MACRO;
  /***************************************************/
  /*                                                 */
  /*  Licensed Materials - Property of IBM           */
  /*                                                 */
  /*  5688-198 (C) Copyright IBM Corp. 1993, 1995    */
  /*  All rights reserved                            */
  /*                                                 */
  /*  US Government Users Restricted Rights - Use,   */
  /*  duplication or disclosure restricted by GSA    */
  /*  ADP Schedule Contract with IBM Corp.           */
  /*                                                 */
  /***************************************************/
 /*Module/File Name: IBM3ABD
 /****************************************************/
 /**                                                 */
 /** Function: CEE3ABD - terminate enclave with an   */
 /**                    abend                        */
 /**                                                 */
 /** In this example, CEE3ABD is called with a       */
 /** timing value of 1.  This requests an abend that */
 /** is deferred until clean-up takes place.         */
 /**                                                 */
 /****************************************************/
 PLI3ABD: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL ABDCODE REAL FIXED BINARY(31,0);
    DCL TIMING  REAL FIXED BINARY(31,0);

    ABDCODE = 3333; /* Choose code to abend with     */
    TIMING = 1;     /* Specify 1, for an abend with  */
                    /*    clean-up                   */

    /* Call CEE3ABD to request an abend 3333 with    */
    /*    clean-up                                   */
    CALL CEE3ABD ( ABDCODE, TIMING );

 END PLI3ABD;
```

CEE3AB2—Terminate enclave with an abend and reason code

CEE3AB2 requests that Language Environment terminate the enclave with an abend and user-defined reason code. The issuing of the abend can be either with or without clean-up, with the type of dumps the user requires. There is no return from this service, nor is there any condition that is associated with it.

Syntax

```
▶▶──CEE3AB2──(──abcode──,──reasoncode──,──clean-up──)──────────────────────▶◀
```

abcode **(input)**
: A fullword integer, no greater than 4095, specifying the abend code that is issued. Under CICS, this fullword integer is converted to EBCDIC.

reasoncode **(input)**
: A fullword integer, specifying the reason code that is issued by the user. If no reason code is specified, the reason code is 0.

clean-up **(input)**
: Indicates if the abend should result in clean-up of the enclave's resources. The acceptable values for *clean-up* are as follows:

0	Issue the abend without clean-up
1	Issue the abend with normal enclave termination processing
2	Issue the abend with enclave termination processing honoring the TERMTHDACT runtime option for taking a system dump of the user address space but suppressing the CEEDUMP
3	Issue the abend with enclave termination processing suppressing both the CEEDUMP and system dump if requested by the TERMTHDACT runtime option
4	Issue the abend with enclave termination processing honoring the TERMTHDACT runtime option for taking a CEEDUMP but suppressing the system dump
5	Issue the abend with enclave termination processing forcing a system dump of the user address space and suppressing the CEEDUMP

If an illegal value for *clean-up* is passed, the abend is issued without clean-up.

If *clean-up* is 0, no Language Environment dump is generated. A system dump, however, is requested when issuing the abend. Under CICS, a transaction dump is taken. To get a dump under CMS, specify FILEDEF SYSABEND PRINTER or FILEDEF SYSUDUMP PRINTER.

If *clean-up* is 0, Language Environment condition handling is disabled for the current enclave and termination activities are not performed. Event handlers are not driven; Debug Tool is not invoked; user exits are not invoked; and user-written condition handlers are not invoked.

When *clean-up* is 1, the abend is processed in the same manner as if it were a non-Language Environment abend. Its processing is affected by the ABPERC and TRAP options, the filedef abends percolated in the assembler user exit, and other elements of the environment related to abend processing. In particular, the condition handler can intercept the abend and give the application a chance to handle the abend. If the condition remains unhandled, normal termination activities are performed: information such as a Language Environment dump is produced, depending on the setting of the TERMTHDACT option; event handlers are driven; Debug Tool is invoked; and user exits are invoked. Assembler user exit settings control if the application actually terminates with an abend.

When *clean-up* is 2, it follows the abend processing as when *clean-up* is 1 except the CEEDUMP is suppressed.

When *clean-up* is 3, it follows the abend processing as when *clean-up* is 1, however CEEDUMP and system dumps are both suppressed

When *clean-up* is 4, it follows the abend processing as when *clean-up* is 1, however only the system dumps are suppressed.

When *clean-up* is 5, it follows the abend processing as when *clean-up* is 1, however forcing a system dump to be taken and suppressing the CEEDUMP.

z/OS UNIX consideration

- In a multithreaded environment, CEE3AB2 applies to the enclave.

Usage notes

1. Language Environment abend codes are usually in the range of 4000 (X'FA0') to 4095 (X'FFF'). Use the range of 0 to 3999 to avoid confusion with Language Environment abend codes. The value that is specified for the abend code is passed directly to the ABEND macro without further verification.
2. When TRAP(OFF) is specified, CEE3AB2 behaves in a similar manner to `clean-up 0`.
3. In a non-CICS environment, a system dump of the user address space is taken only if a SYSMDUMP, SYSUDUMP, or SYSABEND DD card is present.

For more information

- For more information about the ABPERC runtime option, see "ABPERC" on page 9.
- For more information about the TRAP runtime option, see "TRAP" on page 96.
- For more information about the TERMTHDACT runtime option, see "TERMTHDACT" on page 82.
- For more information about the ABEND macro, see *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*.
- For more information about the CEE3ABD callable service, see "CEE3ABD—Terminate enclave with an abend" on page 123.

Examples

1. Following is an example of CEE3AB2 called by C/C++.

```
/*Module/File Name: EDC3AB2    */
/*******************************************************************/
/*                                                                 */
/* THIS EXAMPLE CALLS CEE3AB2 TO TERMINATE THE ENCLAVE WITH AN ABEND */
/* AFTER CLEAN-UP TAKES PLACE. USER COULD ALSO SPECIFY THEIR OWN     */
/* REASON CODE.                                                    */
/*                                                                 */
/*******************************************************************/

#include <leawi.h>

int main(void) {

 _INT4 code,reason,cleanup;
  code = 1234; /* Abend code to issue */
  reason = 9;  /* User defined reason code*/
  cleanup = 3; /* Specify 3, for an ABEND with cleanup & with no dumps*/
  CEE3AB2(&code,&reason,&cleanup);
}
```

2. Following is an example of CEE3AB2 called by COBOL.

```
CBL LIB,QUOTE
      ***********************************************
      *Module/File Name: IGZTAB2
      ***********************************************
      ** Function: CEE3AB2 - Terminate enclave    **
      **           with an abend and user defined **
      **           reason code                    **
      **                                          **
      ** In this example, CEE3AB2 is called to    **
      ** terminate the enclave with an abend and  **
      ** a user defined reason code along with    **
      ** Language Environment cleanup             **
      ***********************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3AB2.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ABDCODE                PIC S9(9) BINARY.
       01  RESCODE                PIC S9(9) BINARY.
       01  TIMING                 PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLMGET.

      ***********************************************
      ** 1234 is the abend code to be issued,     **
      ** 9 is the reason code for the abend.      **
      ** A timing of four requests an abend       **
      ** with clean-up.                           **
      ***********************************************
                MOVE 1234 TO ABDCODE.
                MOVE 9 TO RESCODE.
                MOVE 4 TO TIMING.
                CALL "CEE3AB2" USING ABDCODE, RESCODE, TIMING.

                GOBACK.
```

3. Following is an example of CEE3AB2 called by PL/I.

```
*PROCESS MACRO;

  /*************************************************/
   /*Module/File Name: IBM3AB2                                        */
   /******************************************************************/
   /**                                                              **/
   /** Function: CEE3AB2 - Terminate enclave with abend and user     **/
   /**                    defined reason code                       **/
   /**                                                              **/
   /** In this example, CEE3AB2 is called to terminate the enclave   **/
   /** with an abend and user defined reason code along with         **/
   /** Language Environment cleanup                                  **/
   /**                                                              **/
   /******************************************************************/
  PLI3AB2: PROCEDURE OPTIONS (MAIN) REORDER;

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL ABDCODE REAL FIXED BINARY(31,0);
    DCL RESCODE REAL FIXED BINARY(31,0);
    DCL TIMING  REAL FIXED BINARY(31,0);

    ABDCODE = 3333; /* Choose code to abend with     */
    RESCODE = 9;    /* User defined reason code      */
    TIMING = 4;     /* Specify 4, for an abend with  */
                    /* cleanup and no dumps          */

    /***********************************************/
    /* Call CEE3AB2 to request an abend 3333         */
    /* reason code 9 with cleanup                   */
    /***********************************************/
    CALL CEE3AB2 ( ABDCODE, RESCODE, TIMING );

  END PLI3AB2;
```

CEE3CIB—Return pointer to condition information block

CEE3CIB returns a pointer to a condition information block (CIB) associated with a given condition token. Use this service only during condition handling.

For PL/I and COBOL applications, Language Environment provides called header, COPY, or include files (in SCEESAMP) that map the CIB. For C/C++ applications, the macros are in the header file `leawi.h` (in SCEEH.H).

The CIB contains detailed information about the condition and provides input to your application's condition handlers, which can then respond more effectively to a given condition.

Syntax

```
►►──CEE3CIB──(─┬────────────┬─,─cib_ptr─,─fc─)──────────────────►◄
               └─cond_token─┘
```

cond_token
: The condition token passed to a user-written condition handler. If you do not specify this parameter, Language Environment returns the address of the most recently-raised condition.

cib_ptr
: The address of the CIB associated with the condition token.

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

  The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE35S	1	3260	No condition was active when a call to a condition management routine was made.
CEE35U	1	3262	An invalid condition token was passed. The condition token did not represent an active condition.

Usage notes

- Because the CIB is used only for synchronous signals, you should not use CEE3CIB in signal catchers that are driven for asynchronous signals.
- After the condition handling functions return control to your application, *cib_ptr* is no longer valid.

- z/OS UNIX consideration—In multithread applications, CEE3CIB returns the CIB associated with the current token on only the current thread.

For more information

- For more information about the CEE3CIB callable service, see *z/OS Language Environment Programming Guide*.

Examples

1. Following is example of CEE3CIB called by C/C++.

```
/*Module/File Name: EDC3CIB   */

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>

#ifdef __cplusplus
   extern "C" {
#endif
   void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
   }
#endif

int main(void) {

  _FEEDBACK fc;
  _ENTRY routine;
  _INT4 token;
  int x,y,z;

  /* set the routine structure to point to the handler */
  /* and use CEEHDLR to register the user handler      */

  token = 99;
  routine.address = (_POINTER)&handler;;
  routine.nesting = NULL;

  CEEHDLR(&routine,&token,&fc);
  /* verify that CEEHDLR was successful */
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEEHDLR failed with message number %d\n",
           fc.tok_msgno);
    exit(2999);
  }
  x = 5;
  y = 0;
  z = x / y;
}

/*******************************************************/
/* handler is a user condition handler                 */
/*******************************************************/
void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _CEECIB *cib_ptr;
   _FEEDBACK cibfc;

   CEE3CIB(fc, &cib_ptr, &cibfc);

  /* verify that CEE3CIB was successful */
  if ( _FBCHECK ( cibfc , CEE000 ) != 0 ) {
     printf("CEE3CIB failed with message number %d\n",
            cibfc.tok_msgno);
     exit(2999);
   }

    printf("%s \n",(*cib_ptr).cib_eye);
    printf("%d \n",cib_ptr->cib_cond.tok_msgno);
    printf("%s \n",cib_ptr->cib_cond.tok_facid);
    *result = 10;
 }
```

2. Following is an example of CEE3CIB called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3CIB
      ************************************************
      **                                            **
      ** CBL3CIB - Register a condition handler     **
      **           that will call CEE3CIB to get    **
      **           a Condition Information Block    **
      **           (CIB) for the condition.         **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3CIB.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE  PROCEDURE-POINTER.
       01  TOKEN                  PIC S9(9) BINARY.
       01  FC                     PIC X(12).
       PROCEDURE DIVISION.
       PARA-CBL3CIB.
           SET ROUTINE TO ENTRY "HANDLER".
           CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.

           GOBACK.
       END PROGRAM CBL3CIB.

   CBL LIB,QUOTE
       IDENTIFICATION DIVISION.
       PROGRAM-ID. HANDLER.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  CIB-PTR POINTER.
       01  FC                     PIC S9(9) BINARY.
       LINKAGE SECTION.
      *************************************************
      ** Include the mapping of the CIB             **
      *************************************************
       COPY CEEIGZCI.
       01  CURCOND                PIC X(12).
       01  TOKEN                  PIC S9(9) BINARY.
       01  RESULT                 PIC S9(9) BINARY.
       01  NEWCOND                PIC X(12).
       PROCEDURE DIVISION USING CURCOND, TOKEN,
                                 RESULT, NEWCOND.
       PARA-HANDLER.
           CALL "CEE3CIB" USING CURCOND, CIB-PTR, FC.
           SET ADDRESS OF CEECIB TO CIB-PTR.
           DISPLAY "In Handler".
           DISPLAY CIB-EYE.
           DISPLAY CIB-TOK-MSGNO.
           DISPLAY CIB-TOK-FACID.

           GOBACK.
       END PROGRAM HANDLER.
```

3. Following is an example of CEE3CIB called by PL/I.

```
*PROCESS OPT(0), MACRO;
 /* Module/File Name: IBM3CIB                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3CIB - example of CEE3CIB         **/
 /**                    invoked from PL/I ON-unit  **/
 /**                                                **/
 /****************************************************/

 IBM3CIB:  PROCEDURE  OPTIONS(MAIN);

   %INCLUDE CEEIBMAW;
   %INCLUDE CEEIBMCT;
   %INCLUDE CEEIBMCI;

   DECLARE
      CIB_PTR     POINTER,
      01 FC,                       /* Feedback token */
         03 MsgSev    REAL FIXED BINARY(15,0),
         03 MsgNo     REAL FIXED BINARY(15,0),
         03 Flags,
            05 Case      BIT(2),
            05 Severity  BIT(3),
            05 Control   BIT(3),
         03 FacID     CHAR(3),        /* Facility ID */
         03 ISI     /* Instance-Specific Information */
                 REAL FIXED BINARY(31,0),
      divisor     FIXED BINARY(31) INITIAL(0);

   ON  ZERODIVIDE  BEGIN;

      CALL CEE3CIB(*, CIB_PTR, FC);
      IF  FBCHECK( FC, CEE000 )  THEN  DO;
         PUT SKIP LIST('Found ' || CIB_PTR->CIB_EYE ||
            ' for message #' || CIB_PTR->CIB_TOK_MSGNO
            || ' from ' || CIB_PTR->CIB_TOK_FACID );
         END;
      ELSE  DO;
         DISPLAY( 'CEE3CIB failed with msg '
            || FC.MsgNo );
         END;

      END /* ON ZeroDivide */;

   divisor = 15 / divisor  /* signals ZERODIVIDE */;

 END IBM3CIB;
```

CEE3CTY—Set default country

CEE3CTY sets the default country. A calling routine can change or query the current national country setting. The country setting affects the date format, the time format, the currency symbol, the decimal separator, and the thousands separator.

The current national country setting also affects the format of the date and time in the reports generated by the RPTOPTS and RPTSTG runtime options.

CEE3CTY also affects the default symbols for the NLS and date and time callable services.

CEE3CTY affects only the Language Environment NLS and date and time services, not the Language Environment locale callable services or C locale-sensitive functions.

The current default country setting, as well as previous default settings that have not been discarded, are stored on the stack on a LIFO (last in, first out) basis. The current default country setting is always on the top of the stack.

There are two methods of changing the default country setting with CEE3CTY:

- Specify the new default country setting and place it on top of the stack using a *function* value of 1 (SET). This discards the previous default setting.
- Specify the new default country setting and place it on top of the stack using a *function* value of 3 (PUSH). This pushes the previous default setting down on the stack so that you can later retrieve it by discarding the current setting.

To illustrate the second method, suppose you live in the United States and the code for the United States is specified as the default at installation. If you want to use the French defaults for a certain application, you can use CEE3CTY to PUSH France as the default country setting; then when you want the defaults for the United States, you can POP France from the top of the stack, making the United States the default setting.

Syntax

```
►►──CEE3CTY──(──function──,──country_code──,──fc──)─────────────────────────►◄
```

function **(input)**
: A fullword binary integer specifying the service to be performed. The possible values for *function* are:

1—SET
: Establishes the *country_code* parameter as the current country. The top of the stack is, in effect, replaced with *country_code*.

2—QUERY
: Returns the current country code on the top of the stack to the calling routine. The current code is returned in the *country_code* parameter.

3—PUSH
: Pushes the *country_code* parameter onto the top of the country code stack, making it the current country code. Previous country codes on the stack are retained on a LIFO basis, which makes it possible to return to a prior country code at a later time.

4—POP
: Pops the current country code. The last country code that was affected by a PUSH now becomes the current *country_code*. On return to the calling routine, the *country_code* parameter contains the discarded country code. If the stack contains only one country code, the code cannot be popped because the stack would be empty after the call. Therefore, no action is taken and a feedback code indicating such is returned to the calling routine.

`country_code` **(input/output)**

A 2-character fixed-length string. *country_code* is not case-sensitive. Table 32 on page 481 contains a list of valid country codes. It is used in the following ways for the different *functions:*

If *function* is:	then *country_code*:
1 or 3	Contains the desired 2-character country code. In this case, it is an input parameter. Table 32 on page 481 contains a list of valid country codes.
2	Returns the current 2-character country code on top of the stack. In this case, it is an output parameter.
4	Returns the discarded 2-character country code. In this case, it is an output parameter.

`fc` **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3BV	2	3455	Only one country code was on the stack when a POP request was made to CEE3CTY. The current country code was returned in the
CEE3C0	3	3456	The country code *country-code* for the PUSH or SET function for CEE3CTY was invalid. No operation was performed.
CEE3C1	3	3457	The function *function* specified for CEE3CTY was not recognized. No operation was performed.

Usage notes

- The default setting for *country_code* is already on the stack when you start a program. You do not have to PUSH it there.
- The bytes X'0E' and X'0F' representing shift-out and shift-in codes are not affected by any *country_code* setting.
- C/C++ consideration—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 CEE3CTY callable service. CEE3CTY 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 CEE3CTY and either CEESETL or `setlocale()`.
- z/OS UNIX consideration—CEE3CTY applies to the enclave. Every thread in the enclave has the same country setting.

For more information

- For more information about the RPTOPTS and RPTSTG runtime options, see "RPTOPTS" on page 69 and "RPTSTG" on page 71.
- For a list of the default settings for a specified country, see Table 32 on page 481.
- For more information about the COUNTRY runtime option, see "COUNTRY" on page 22.
- For more information about `setlocale()`, see *z/OS XL C/C++ Programming Guide*.

Examples

1. Following is an example of CEE3CTY called by C/C++.

```
/*Module/File Name: EDC3CTY   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _INT4 function;
  _CHAR2 country;

  /* query the current country setting */
  function = 2; /* function 2 is query */
  CEE3CTY(&function,country,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE3CTY failed with message number %d\n",
             fc.tok_msgno);
     exit(2999);
  }

  /* if the current country is not Canada then set */
  /* it to Canada */
  if (memcmp(country,"CA",2) != 0) {
     memcpy(country,"CA",2);
     function = 1; /* function 1 is set */
     CEE3CTY(&function,country,&fc);

     if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEE3CTY failed with message number %d\n",
               fc.tok_msgno);
        exit(2999);
     }
  }
}
```

2. Following is is an example of CEE3CTY called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3CTY
      ************************************************
      **                                            **
      ** Function: CEE3CTY - set default country    **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3CTY.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  FUNCTN                 PIC S9(9) BINARY.
       01  COUNTRY                PIC X(2).
```

```
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.
 PARA-3CTYQRY.
************************************************
** Call CEE3CTY with the QUERY function,
**     and display current country code.
************************************************
     MOVE 2 TO FUNCTN.
     CALL "CEE3CTY" USING FUNCTN, COUNTRY, FC.
     IF CEE000 of FC  THEN
         DISPLAY "THE CURRENT COUNTRY CODE IS "
             COUNTRY
     ELSE
         DISPLAY "CEE3CTY(query) failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

 PARA-3CTYSET.
************************************************
** If the current country code is not the US,
** call CEE3CTY with the SET function to make
** it the US. Display result.
************************************************
     IF ( COUNTRY IS NOT = "US" ) THEN
         MOVE 1 TO FUNCTN
         MOVE "US" TO COUNTRY
         CALL "CEE3CTY" USING FUNCTN,
                              COUNTRY, FC
         IF CEE000 of FC  THEN
             DISPLAY "THE NEW COUNTRY CODE IS ",
                 COUNTRY
         ELSE
             DISPLAY "CEE3CTY(set) failed with msg "
                 Msg-No of FC UPON CONSOLE
             STOP RUN
         END-IF
     END-IF.

     GOBACK.
```

3. Following is an example of CEE3CTY called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3CTY                        */
 /***************************************************/
 /**                                               **/
 /** Function: CEE3CTY - set current country       **/
 /**                                               **/
 /** In this example, a call is made to the query  **/
 /** function of CEE3CTY to return the current     **/
 /** default country setting.  This is then        **/
 /** printed out. If the current country code is   **/
 /** not 'US', then it is set to 'US' and printed. **/
 /**                                               **/
```

```
/****************************************************/
PLI3CTY: PROC OPTIONS(MAIN);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL FUNCTN   REAL FIXED BINARY(31,0);
   DCL COUNTRY  CHARACTER ( 2 );
   DCL 01 FC,                    /* Feedback token */
          03 MsgSev    REAL FIXED BINARY(15,0),
          03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),      /* Facility ID */
          03 ISI   /* Instance-Specific Information */
                     REAL FIXED BINARY(31,0);

   FUNCTN = 2; /* Specify 2 for the query function  */

   /* Call CEE3CTY with the query function to       */
   /*    return the current country setting         */
   CALL CEE3CTY ( FUNCTN, COUNTRY, FC );

   /* If CEE3CTY ran successfully, print the        */
   /*    current country                            */
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST( 'The current country code is "'
         || COUNTRY || '".' );
      END;
   ELSE  DO;
      DISPLAY( 'CEE3CTY failed with msg '
         || FC.MsgNo );
      STOP;
      END;/* If the current default country is not the US, */
   /*    set it to the US                           */
   IF  COUNTRY ¬= 'US'  THEN DO;
      FUNCTN = 1; /* Specify 1 for the set function */
      COUNTRY = 'US'; /* Specify country code for US*/
      CALL CEE3CTY ( FUNCTN, COUNTRY, FC );

      /* If CEE3CTY ran successfully print the      */
      /*    current country                         */
      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST( 'The new country code is "'
            || COUNTRY || '".' );
         END;
      ELSE  DO;
         DISPLAY( 'CEE3CTY failed with msg '
            || FC.MsgNo );
         STOP;
         END;

      END;

END PLI3CTY;
```

CEE3DLY—Suspend processing of the active enclave in seconds

CEE3DLY provides a service for Language Environment-conforming applications that suspends the processing of the active enclave for a specified number of seconds. The maximum is 1 hour.

Syntax

```
►►—CEE3DLY—(—input_seconds—,—fc—)——————————————————————————————►◄
```

input_seconds
A fullword binary value in the range of 0 to 3600 that specifies the total number of seconds during which the enclave should be suspended.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE3QQ	1	CEE3930W	The input value *input_value* in a call to the callable service *callable_service_name* was not within the valid range.
CEE3QS	1	CEE3932W	The system service *system_service* failed with return code *return_code* and reason code *reason_code*.

Usage notes

- CICS consideration—CEE3DLY is available under CICS.
- z/OS UNIX consideration—CEE3DLY is handled by the z/OS UNIX System Services when POSIX is set to ON.
- This service is not intended for timing requests. Delays up to the nearest second might occur in some circumstances.
- In a multi-threaded application, only the calling thread is suspended.

Examples

1. An example of CEE3DLY called by C/C++:

```
/*Module/File Name: EDC3DLY   */
/*********************************************************************/
/*                                                                   */
/* THIS EXAMPLE CALLS CEE3DLY TO SUSPEND PROCESSING OF THE ACTIVE    */
/* ENCLAVE FOR SPECIFIED NUMBER OF SECONDS.                          */
/*                                                                   */
/*********************************************************************/
#include <time.h>
#include <stdio.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void)
{
  _INT4 seconds;
  _FEEDBACK fc;
  time_t ltime;

  /* Get the time in seconds */
  time(&ltime);
  printf("CEE3DLY Start  time : %s", ctime(&ltime));

  seconds = 10;
  CEE3DLY(&seconds,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEE3DLY failed with message number %d\n",fc.tok_msgno);
    exit(999);
  }

  time(&ltime);
  printf("CEE3DLY Finish time : %s", ctime(&ltime));
}
```

2. A example of CEE3DLY called by COBOL:

```
CBL LIB,QUOTE
      *****************************************************************
      *MODULE/FILE NAME: IGZT3DLY
      *****************************************************************
      **                                                             **
      ** FUNCTION: CEE3DLY - SUSPEND ENCLAVE EXECUTION IN SECONDS    **
      **                                                             **
      ** THIS EXAMPLE CALLS CEE3DLY TO SUSPEND PROCESSING OF THE     **
      ** ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF SECONDS.             **
      **                                                             **
      *****************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3DLY.
       DATA DIVISION.  WORKING-STORAGE SECTION.
       01  DATE-TIME                    PIC X(21).
       01  CUR-DATE-FIELDS.
           03  CURRENT-DATE             PIC X(8).
           03  FILLER REDEFINES CURRENT-DATE.
               05  CUR-MM               PIC XX.
               05                       PIC X.
               05  CUR-DD               PIC XX.
               05                       PIC X.
               05  CUR-YY               PIC XX.
           03  TIME-OF-DAY              PIC X(8).
       01  SECONDS   PIC S9(9) BINARY.
       01  FC.
           02  CONDITION-TOKEN-VALUE.
           COPY  CEEIGZCT.
               03  CASE-1-CONDITION-ID.
                  04  SEVERITY     PIC S9(4) BINARY.
                  04  MSG-NO       PIC S9(4) BINARY.
               03  CASE-2-CONDITION-ID
                        REDEFINES CASE-1-CONDITION-ID.
                  04  CLASS-CODE   PIC S9(4) BINARY.
                  04  CAUSE-CODE   PIC S9(4) BINARY.
               03  CASE-SEV-CTL    PIC X.
               03  FACILITY-ID     PIC XXX.
           02  I-S-INFO            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBL3DLY.
           MOVE FUNCTION CURRENT-DATE TO DATE-TIME.
           STRING DATE-TIME (5:2) "/"
                  DATE-TIME (7:2) "/"
                  DATE-TIME (3:2)
                      DELIMITED BY SIZE
                      INTO CURRENT-DATE.
           STRING DATE-TIME (9:2)  ":"
                  DATE-TIME (11:2) ":"
                  DATE-TIME (13:2)
                      DELIMITED BY SIZE
                      INTO TIME-OF-DAY.
           DISPLAY "CEE3DLY - BEGINS"
           DISPLAY "DATE    : " CURRENT-DATE
           DISPLAY "TIME    : " TIME-OF-DAY
           MOVE 10 TO SECONDS.
           CALL "CEE3DLY" USING SECONDS, FC.
           IF NOT CEE000 OF FC THEN
              DISPLAY "CEE3DLY FAILED WITH MSG "
                       MSG-NO OF FC
              STOP RUN
           END-IF.
           MOVE FUNCTION CURRENT-DATE TO DATE-TIME.
           STRING DATE-TIME (5:2) "/"
                  DATE-TIME (7:2) "/"
                  DATE-TIME (3:2)
                      DELIMITED BY SIZE
                      INTO CURRENT-DATE.
           STRING DATE-TIME (9:2)  ":"
                  DATE-TIME (11:2) ":"
                  DATE-TIME (13:2)
                      DELIMITED BY SIZE
                      INTO TIME-OF-DAY.
           DISPLAY "CEE3DLY - COMPLETED"
           DISPLAY "DATE    : " CURRENT-DATE
           DISPLAY "TIME    : " TIME-OF-DAY
           GOBACK.
```

3. An example of CEE3DLY called by PL/I:

```
*PROCESS MACRO;
    /***************************************************/
  /*MODULE/FILE NAME: IBM3DLY                                       */
  /******************************************************************/
  /**                                                             **/
  /** FUNCTION: CEE3DLY - SUSPENDS ENCLAVE EXECUTION IN SECONDS     **/
  /**                                                             **/
  /** THIS EXAMPLE CALLS CEE3DLY TO SUSPEND PROCESSING OF THE       **/
  /** ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF SECONDS.               **/
  /**                                                             **/
  /******************************************************************/
  PLI3DLY: PROCEDURE OPTIONS (MAIN) REORDER;

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DECLARE DLYSECS  REAL FIXED BINARY(31,0);
    DECLARE LILIAN   REAL FIXED BINARY(31,0);
    DECLARE SECONDS  REAL FLOAT DECIMAL(16);
    DECLARE GREGORN  CHARACTER (17);

    DECLARE 01 FC1,            /* FEEDBACK TOKEN FOR CEELOCT */
               03 MSGSEV       REAL FIXED BINARY(15,0),
               03 MSGNO        REAL FIXED BINARY(15,0),
               03 FLAGS,
                  05 CASE      BIT(2),
                  05 SEVERITY  BIT(3),
                  05 CONTROL   BIT(3),
               03 FACID        CHAR(3),
               03 ISI          REAL FIXED BINARY(31,0);

    DECLARE 01 FC2,            /* FEEDBACK TOKEN FOR CEE3DLY */
               03 MSGSEV       REAL FIXED BINARY(15,0),
               03 MSGNO        REAL FIXED BINARY(15,0),
               03 FLAGS,
                  05 CASE      BIT(2),
                  05 SEVERITY  BIT(3),
                  05 CONTROL   BIT(3),
               03 FACID        CHAR(3),
               03 ISI          REAL FIXED BINARY(31,0);

    CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC1 );
    IF FBCHECK(FC1, CEE000)  THEN  DO;
      PUT SKIP LIST ( 'CEE3DLY START  DATE AND TIME: ' || GREGORN );
  END;
  ELSE DO;
    PUT ( 'CEELOCT FAILED WITH MSG ' || FC1.MSGNO );
    STOP;
  END;
 DLYSECS = 30;
  CALL CEE3DLY(DLYSECS, FC2);
  IF FBCHECK(FC2, CEE000) THEN DO;
    CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC1 );
    IF  FBCHECK(FC1, CEE000)  THEN  DO;
      PUT SKIP LIST ( 'CEE3DLY FINISH DATE AND TIME: ' || GREGORN );
    END;
    ELSE DO;
      PUT ('CEELOCT FAILED WITH MSG ' || FC1.MSGNO );
      STOP;
    END;
    PUT SKIP LIST ( 'CEE3DLY IS SUCCESSFUL!' );
  END;
  ELSE DO;
    PUT SKIP LIST ( 'CEE3DLY FAILED WITH MSG ' || FC2.MSGNO );
    STOP;
  END;

END PLI3DLY;
```

CEE3DMP—Generate dump

CEE3DMP generates a dump of Language Environment and the member language libraries. Sections of the dump are selectively included, depending on options specified with the *options* parameter. When an error occurs that would cause a CEEDUMP to be taken, and this is a POSIX application, Language Environment writes this dump to the current directory. Output from CEE3DMP is written to one of the following (in top-down order):

- The directory found in _CEE_DMPTARG, if found.
- The current working directory, if this is not the root (/), and if the directory is writable, and if the dump pathname (made up of the cwd pathname plus the dump file name) does not exceed 1024 characters.
- The directory found in environment variable TMPDIR (if the temporary directory is not /TMP.
- /TMP

(See the Fname suboption of CEE3DMP in 147 for more information about the dump output filename.)

If longer than 60 characters, the dump title is truncated to 60 characters in order to match the record size of the dump file. Only nested enclaves within a single process are supported.

CEE3DMP establishes a condition handler that captures all conditions that occur during dump processing. It terminates the section of the dump in progress when a condition occurs and inserts the following line into the dump (*nnnnnnnn* is the instruction address at the time of the exception). After this line is inserted in the report, dump processing continues for other member languages until CEE3DMP is complete.

```
Exception occurred during dump processing at nnnnnnnn
```

If an abend occurs, or if any other condition occurs or is raised, the condition manager attempts to handle it. If the condition remains unhandled, and it is of sufficient severity, the condition manager might, based on if the TERMTHDACT TRACE or DUMP suboption is specified, invoke dump services and then terminate the program. You do not have to call CEE3DMP to use the dump services. Any routine can use them. To support this case, dump services are invoked as follows:

- The title is `Condition processing resulted in the unhandled condition` to indicate why the dump was produced.
- The dump options are 'TRACE COND THR(ALL) BLOCKS STOR NOENTRY' for dump output and 'TRACE COND THR(ALL) NOBLOCK NOSTOR NOENTRY' for trace output.

Reprinting of section title and control block name at the top of each page is suppressed. Only the main title `CEE3DMP: ...` is reprinted.

Syntax

```
►►──CEE3DMP──(──title──,──options──,──fc──)──────────────────────────►◄
```

title **(input)**
: An 80-byte fixed-length character string containing a title printed at the top of each page of the dump.

options
: A 255-byte fixed-length character string enclosed in single quotes containing options describing the type, format, and destination of dump information. Options are declared as a string of keywords separated by blanks or commas. Some options have suboptions that follow the option keyword and are contained in parentheses. The options can be specified in any order, but the last option declaration is honored if there is a conflict between it and any preceding options. The following options are recognized by Language Environment:

ENCLave(ALL | CURrent | *n*)
: Dumps the current enclave, a fixed number of enclaves, or all enclaves associated with the current process; *n* is an integer ranging from 1 to 2**31-1, inclusive, that indicates the maximum number of enclaves that should be dumped. ENCLAVE(CURRENT) and ENCLAVE(1) are equivalent.

THRead(ALL|CURrent)
: Dumps the current thread (the thread that invoked this service) or all threads associated with the current enclave. When you specify THREAD(ALL) and more than one thread is running, the library quiesces all threads before writing the dump. Therefore, the state of the library changes from the time the dump is requested to the time the dump is written.

TRACEback
: Includes a traceback of all routines on the call chain. The traceback shows transfers of control from either calls or exceptions. The traceback extends backwards to the main program of the current thread. PL/I transfers of control into BEGIN-END blocks or ON-units are considered calls.

NOTRACEback
: Does not include a traceback.

FILEs
: Includes attributes of all open files and the buffer contents used by the files. The particular attributes displayed are defined by the member languages. File buffers are dumped when FILE and STORAGE are specified. File control blocks are dumped when FILE and BLOCKS are specified.

NOFILEs
: Does not include file attributes of open files.

VARiables
: Includes a symbolic dump of all variables, arguments, and registers. Variables include arrays and structures. Register values are those saved in the stack frame at the time of call. There is no way to print a subset of this information. Variables and arguments are printed only if the symbol tables are available. A symbol table is generated when a program is compiled with the options shown below for each HLL, except for PL/I, which does not support the VARIABLE option.

Language	Compile-Time Option
C	TEST(SYM)
C++	TEST
COBOL	TEST or TEST(h,SYM)

The variables, arguments, and registers are dumped, beginning with the routine that called CEE3DMP. The dump proceeds up the chain for the number of routines specified by the STACKFRAME option. See below for a description of the STACKFRAME option.

NOVARiables
: Does not include a dump of variables, arguments, and registers.

BLOCKs
: Dumps the control blocks used in Language Environment and member language libraries. Global control blocks, as well as control blocks associated with routines on the call chain, are printed. Control blocks are printed for the routine that called CEE3DMP. The dump proceeds up the call chain for the number of routines specified by the STACKFRAME option (see below). Control blocks for files are also dumped if the FILES option was specified. See the FILES option above for more information. If the TRACE runtime option is set to ON, the trace table is dumped when BLOCKS is specified. If the heap storage diagnostics report is requested through the HEAPCHK runtime option, the report is displayed when BLOCKS is specified.

NOBLOCKs
: Suppresses the dump of control blocks.

STORage
: Dumps the storage used by the program. The storage is displayed in hexadecimal and character format. Global storage, as well as storage associated with each routine on the call chain, is printed. Storage is dumped for the routine that called CEE3DMP, which proceeds up the call

chain for the number of routines specified by the STACKFRAME option. Storage for all file buffers is also dumped if the FILES option was specified (see above).

NOSTORage
: Suppresses storage dumps.

REGSTor
: Controls the amount of storage (`reg_stor_amount`) that is dumped around registers. `reg_stor_amount` indicates storage in bytes to be dumped around each register and must be in the range of 0 to 256. The number is rounded up to the nearest multiple of 32. The default is REGSTOR(96).

  Restriction: You must specify REGSTor(0) as a CEE3DMP option if a dump storage around registers is not required.

StackFrame(*n*|ALL)
: Specifies the number of stack frames dumped from the call chain. If STACKFRAME(ALL) is specified, all stack frames are dumped. No stack frame storage is dumped if STACKFRAME(0) is specified. The particular information dumped for each stack frame depends on the VARIABLE, BLOCK, and STORAGE option declarations specified for CEE3DMP. The first stack frame dumped is the one associated with the routine that called CEE3DMP, followed by its caller, and proceeding backwards up the call chain.

PAGEsize(*n*)
: Specifies the number of lines on each page of the dump. This value must be greater than 9. A value of 0 indicates that there should be no page breaks in the dump. The default setting is PAGESIZE(60). Refer to the CEEDUMP runtime option for an optional method of setting the number of lines on each page of the dump.

FNAME(*ddname*)
: Specifies the ddname of the file to which the dump report is written. The ddname supplied in this option must be a valid ddname for the system on which the application is running. CEE3DMP does not check the ddname for validity, nor does CEE3DMP translate or modify the ddname. Supplying an invalid ddname can result in unpredictable behavior. The default ddname CEEDUMP is used if this option is not specified.

CONDition
: Specifies that for each active condition on the call chain, the following information is dumped from the CIB:
  - The address of the CIB.
  - The message associated with the current condition token.
  - The message associated with the original condition token, if different from the current one.
  - The location of the error.
  - The machine state at the time the condition manager was invoked, if an abend or hardware condition occurred.
  - The abend code and reason code, if the condition occurred as a result of an abend.
  - Language-specific error information.

    The information supplied by Language Environment-conforming languages differs. PL/I supplies DATAFIELD, ONCHAR, ONCOUNT, ONFILE, ONKEY, and ONSOURCE built-in function (BIF) values, which are shown in the context of the condition raised.

This option does not apply to asynchronous signals.

NOCONDition
Does not dump condition information for active conditions on the call chain.

ENTRY
Includes in the dump a description of the routine that called CEE3DMP and the contents of the registers on entry to CEE3DMP.

NOENTRY
Does not include in the dump a description of the routine that called CEE3DMP and the contents of the registers on entry to CEE3DMP.

GENOpts
Includes in the dump a runtime options report like that generated by the use of the RPTOPTS runtime option.

NOGENOpts
Does not include in the dump a runtime options report like that generated by the use of the RPTOPTS runtime option.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted. If specified as an argument, feedback information, in the form of a condition token, is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

The following feedback codes can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE317	1	3111	CEEDUMP was defined as a dummy data set. No Language Environment dump processing was performed.
CEE30U	2	3102	Invalid CEE3DMP options or suboptions were found and ignored.
CEE30V	3	3103	An error occurred in writing messages to the dump file.

Usage notes

- Language Environment is sensitive to the national language when it writes dump output.
  - When the national language is upper case U.S. English or Japanese, the environment variable _CEE_UPPERCASE_DATA can be used to determine if variable data in the 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, hexadecimal/EBCDIC displays of storage) will be in uppercase.
  - When this environment variable is not set or set to a value other than YES, variable data will not be in uppercase. Variable data is never in uppercase when the national language is mixed-case U.S. English.

- CICS consideration—Only ENCLAVE(CURRENT) and ENCLAVE(1) are supported on CICS.
- MVS consideration—On MVS, all values for the ENCLAVE option are supported.
- z/OS UNIX consideration—CEE3DMP applies to the enclave.

  When you call CEE3DMP in a multithread environment, the current thread or all threads might be dumped. Enclave- and process-related storage (along with storage related to threads other than the current thread) might have changed in value from the time the dump request was issued.

  If the CEEDUMP DD has a PATH= parameter, the dump is directed to the specified file system.

  If your application is running under z/OS UNIX System Services and is either running in an address space created by using the `fork()` function or is invoked by one of the `exec` family of functions, the dump is written to the file system. Language Environment writes the dump to a file in your current working directory, unless that directory is the root directory, in which case the dump is written to a file in the directory `/tmp`..

  The name of this file changes with each dump and uses the following format:

  ```
  /path/Fname.Date.Time.Pid
  ```

path	The current working directory
Fname	The name specified in the FNAME parameter on the call to CEE3DMP (default is CEEDUMP)
Date	The date the dump is taken, appearing in the format `YYYYMMDD` (such as 19940325 for March 25, 1994)
Time	The time the dump is taken, appearing in the format `HHMMSS` (such as 175501 for 05:55:01 PM)
Pid	The process ID the application is running in when the dump is taken

- When nested CEEPIPI main-DP environments are present, the traceback section includes all nested environments. The first environment contains all the usual sections as determined by the CEE3DMP options. For the chained parent main-DP environments, the dump output omits all sections other than the traceback.

  The ENCLAVE option controls how many enclaves appear in the output for each CEEPIPI main-DP environment. The STACKFRAME option controls how many stack frames are dumped for each CEEPIPI main-DP environment.
- When multiple CEEPIPI main-DP environments are present, and the chained parent Main-DP environments contain Language Environment member languages that are not present in the current active Main-DP environment (the one that caused the dump), the information dumped out might be incomplete.
- If the assembler CEEPIPI driver program does not follow standard z/OS linkage conventions, the dumped traceback information might be incomplete or incorrect.

For more information

- See "TERMTHDACT" on page 82 for more information about the TERMTHDACT runtime option.
- For more information about generating dumps, see *z/OS Language Environment Debugging Guide*.

Examples

1. Following is an example of CEE3DMP called by C/C++.

```
/*Module/File Name: EDC3DMP   */

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>

#define OPT_STR "THREAD(CURRENT) TRACEBACK FILES"

int main(void) {

  _CHAR80 title =
           "This is the title of the dump report";
  _CHAR255 options;
  FILE *f;
  _FEEDBACK fc;

  memset(options,' ',sizeof(options));
  memcpy(options,OPT_STR,sizeof(OPT_STR)-1);

  f = fopen("my.file","wb");
  fprintf(f,"my file record 1\n");

  CEE3DMP(title,options,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE3DMP failed with msgno %d\n",
       fc.tok_msgno);
     exit(2999);
  }
}
```

2. Following is an example of CEE3DMP called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3DMP
      *************************************************
      **                                             **
      ** CBL3DMP - Call CEE3DMP to generate a dump   **
      **                                             **
      ** In this example, a call to CEE3DMP is made  **
      ** to request a dump of the runtime          **
      ** environment. Several options are specified  **
      ** to customize the dump.                      **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3DMP.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  DMPTITL                PIC X(80).
       01  OPTIONS                PIC X(255).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info           PIC S9(9) BINARY.
       PROCEDURE DIVISION.
```

```
*************************************************
** Specify title to appear on each page of the
**     dump report.
** Specify options that will request that a
**     traceback be provided, but no variables,
**     stack frames, condition information, or
**     registers be dumped.
*************************************************
 PARA-CBL3DMP.
     MOVE "This is the dump report title."
           TO DMPTITL.
     MOVE "TRACE NOVAR SF(0) NOCOND NOENTRY"
           TO OPTIONS.
     CALL "CEE3DMP" USING DMPTITL, OPTIONS, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEE3DMP failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEE3DMP called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3DMP                          */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3DMP - generate dump              **/
 /**                                                **/
 /** In this example, a call to CEE3DMP is made to  **/
 /** request a dump of the runtime environment.    **/
 /** Several options are specified, to customize    **/
 /** the dump.                                      **/
 /****************************************************/
 PLI3DMP: PROC OPTIONS(MAIN);
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL DMPTITL  CHAR(80);
    DCL OPTIONS  CHARACTER ( 255 );
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    /* Specify a string to be printed at the top of  */
    /*    each page of dump                          */

    DMPTITL = 'This is the title for the dump report.';

    /* Request that a traceback be provided, but     */
    /*    no variables, stack frames, condition      */
    /*    information, or registers be dumped        */

    OPTIONS = 'TRACE NOVAR SF(0) NOCOND NOENTRY';

    /* Call CEE3DMP with options to customize dump   */
    CALL CEE3DMP ( DMPTITL, OPTIONS, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'Successfully produced dump with'
          || ' title "' || DMPTITL || '"' );
```

```
            PUT SKIP LIST( ' and options:  ' || OPTIONS );
            END;
         ELSE  DO;
            DISPLAY( 'CEE3DMP failed with msg '
               || FC.MsgNo );
            STOP;
            END;

      END PLI3DMP;
```

CEE3GRC—Get the enclave return code

CEE3GRC retrieves the current value of the user enclave return code. Use CEE3GRC in conjunction with CEE3SRC to get and then set user enclave return codes.

Syntax

```
▸▸──CEE3GRC──(──return_code──,──fc──)─────────────────────────────────▸◂
```

return_code **(output)**
The enclave return code.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Usage notes

- z/OS UNIX consideration—CEE3GRC is not supported in multithread applications.
- PL/I MTF consideration—CEE3GRC is not supported in PL/I MTF applications.
- PL/I consideration—When running PL/I with POSIX(ON), CEE3GRC is not supported.

For more information

- See "CEE3SRC—Set the enclave return code" on page 206 for more information about the CEE3SRC callable service.
- See *z/OS Language Environment Programming Guide* for more information about the CEE3GRC and CEE3SRC callable services.

Examples

1. Following is an example of a C/C++ main() routine that ralls CEEHDLR and CEE3GRC.

```
/*Module/File Name: EDC3GRC   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

 /*****************************************************/
 /**                                                 */
 /** Function: CEEHDLR - Register user condition     */
 /**                     handler                     */
 /**         : CEE3GRC - Get enclave return code     */
 /*                                                  */
 /* 1. Register the user-written condition handler   */
 /*    CESETRC.                                      */
 /* 2. Call CERCDIV, which performs a divide-by-zero. */
 /* 3. CESETRC is entered, sets the enclave return   */
 /*    code to 999998 and resumes.                   */
 /* 4. The main routine regains control and          */
 /*    retrieves the enclave return code             */
 /*****************************************************/

void CERCDIV(int);
 /****************************************************
   Declaration of user-written condition handler
  ****************************************************/
#ifdef __cplusplus
   extern "C" {
#endif
   void CESETRC(_FEEDBACK *, _INT4*, _INT4 *, _FEEDBACK *);
#ifdef __cplusplus
   }
#endif

main()
{
 _INT4 idivisor = 0;
 _INT4 enclave_RC;
 _FEEDBACK feedback, new_feedback;
 _ENTRY pgmptr;
 _INT4 token;

 /****************************************************
  The condition handler CESETRC is registered
  ****************************************************/
 pgmptr.address = (_POINTER)&CESETRC;;
 pgmptr.nesting = NULL;
 token = 97;
 CEEHDLR(&pgmptr, &token, &feedback);

 /****************************************************
  A divide-by-zero is accomplished by calling CERCDIV.
  ****************************************************/
 CERCDIV(idivisor);   /* this causes a zero divide    */

 /****************************************************
  Call CEE3GRC and check that enclave return code was set.
  ****************************************************/
 CEE3GRC(&enclave_RC, &feedback);
   if ( _FBCHECK ( feedback , CEE000 ) != 0 ) {
      printf("CEE3GRC failed with message number %d\n",
             feedback.tok_msgno);
      exit(2999);
   }
 if (enclave_RC != 999998)
   printf ("Error setting enclave return code");
}
```

2. Following is an example of a C/C++ user-written condition handler that sets a user enclave return code.

```
/*Module/File Name: EDC3SRC   */
 /****************************************************/
 /**                                                 */
 /** Function: CEE3SRC - Set the enclave return code. */
 /*                                                  */
 /* This is the user-written condition handler       */
 /* registered by CEGETRC. It invokes CEE3SRC to set  */
 /* the enclave return code to 999998                */
 /* when a divide-by-zero condition is encountered.   */
 /**                                                 */
 /****************************************************/
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>
#define RESUME 10
#define PERCOLATE 20
 /****************************************************/

#ifdef __cplusplus
 extern "C" void CESETRC (_FEEDBACK *, _INT4 *,
 _INT4 *, _FEEDBACK *);
#endif

void CESETRC (_FEEDBACK *cond, _INT4 *input_token,
              _INT4 *result, _FEEDBACK *new_cond)
{
 _INT4 enclave_RC;
 _FEEDBACK feedback;

 if ( _FBCHECK ( *cond , CEE349 ) == 0 )
  {
   enclave_RC = 999998;
   CEE3SRC(&enclave_RC, &feedback);
   *result = RESUME;
  }
 else
  {
   *result = PERCOLATE;
  }
}
```

3. Following is an example of a C/C++ subroutine that generates the divide-by-zero condition.

```
/*Module/File Name: EDCDIV    */
#include <stdio.h>
#include <string.h>
 /*************************************************/
 /**                                              */
 /* This is a divide-by-zero routine. It divides   */
 /* an input integer by a constant.                */
 /**                                              */
 /*************************************************/

void CERCDIV (int Integer)

{
 int num;
 num = 1/Integer;
}
```

4. Following is an example of a COBOL main routine that calls CEEHDLR and CEE3GRC.

```
CBL LIB,QUOTE,C,RENT,OPTIMIZE,NODYNAM
      *Module/File Name: IGZT3GRC
      *************************************************
      **                                             **
      ** CBL2GRC - Call the following Lang. Env. svcs: **
      **                                             **
      **        : CEEHDLR - register user condition  **
      **                    handler                  **
      **        : CEE3GRC - get enclave return code  **
      **                                             **
      ** 1. Registers user condition handler CESETRC. **
      ** 2. Program then calls CERCDIV which performs **
      **    a divide by zero operation.              **
      ** 3. CESETRC gets control and set the enclave  **
      **    return code to 999998 and resumes.       **
      ** 4. Regains control and retrieves the enclave **
      **    return code.                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    CBL3GRC.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  TOKEN           PIC X(4).
       01  IDIVISOR        PIC S9(9)
                           BINARY VALUE ZERO.
       01  ENCLAVE-RC      PIC S9(9) BINARY.
      **
      **  Declares for condition handling
      **
       01  PGMPTR          USAGE IS PROCEDURE-POINTER.
       01  FBCODE.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.

       PROCEDURE DIVISION.

       0001-BEGIN-PROCESSING.
      *************************************************
      ** Register user condition handler CESETRC using
      **     CEEHDLR
      *************************************************
           SET PGMPTR TO ENTRY  "CESETRC".
           MOVE 97 TO TOKEN
           CALL "CEEHDLR" USING PGMPTR, TOKEN, FBCODE.
           IF NOT CEE000 of FBCODE  THEN
               DISPLAY "CEEHDLR failed with msg "
                   Msg-No of FBCODE UPON CONSOLE
               STOP RUN
           END-IF.

      *************************************************
      ** Call CERCDIV  to cause a divide by zero
      **     condition
      *************************************************
           CALL "CERCDIV" USING IDIVISOR.
      *************************************************
      ** Call CEE3GRC  to get the enclave return code
      *************************************************
           CALL "CEE3GRC" USING ENCLAVE-RC, FBCODE.
           IF NOT CEE000 of FBCODE  THEN
               DISPLAY "CEEHDLR failed with msg "
                   Msg-No of FBCODE UPON CONSOLE
               STOP RUN
           END-IF.

           IF (ENCLAVE-RC =  999998) THEN
               DISPLAY "Enclave return code "
                   "set and retrieved."
           ELSE
               DISPLAY "*** Unexpected enclave return "
                   "code of " ENCLAVE-RC " encountered"
           END-IF.

           GOBACK.
       End program CBL3GRC.
```

5. Following is an example of a COBOL condition handler that sets a user enclave return code and resumes when a divide-by-zero condition occurs.

```
   CBL C,RENT,OPTIMIZE,NODYNAM,LIB,QUOTE
      *Module/File Name: IGZT3SRC
      *************************************************
      **                                             **
      ** DRV3SRC - Drive sample program for CEE3SRC. **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRV3SRC.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE         PROCEDURE-POINTER.
       01  DENOMINATOR     PIC S9(9) BINARY.
       01  NUMERATOR       PIC S9(9) BINARY.
       01  RATIO           PIC S9(9) BINARY.
       01  TOKEN           PIC S9(9) BINARY VALUE 0.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.

       PROCEDURE DIVISION.

       REGISTER-HANDLER.
      *************************************************
      ** Register handler
      *************************************************
           SET ROUTINE TO ENTRY "CBL3SRC".
           CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
           IF  NOT CEE000 of FC  THEN
               DISPLAY "CEEHDLR failed with msg "
                   Msg-No of FC UPON CONSOLE
```

```
        STOP RUN
    END-IF.   RAISE-CONDITION.
*************************************************
** Cause a zero-divide condition.
*************************************************
    MOVE 0 TO DENOMINATOR.
    MOVE 1 TO NUMERATOR.
    DIVIDE NUMERATOR BY DENOMINATOR
           GIVING RATIO.

UNREGISTER-HANDLER.
*************************************************
** UNregister handler
*************************************************
    CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
    IF NOT CEE000 of FC  THEN
        DISPLAY "CEEHDLU failed with msg "
            Msg-No of FC UPON CONSOLE
    END-IF.

    STOP RUN.
 END PROGRAM DRV3SRC.

************************************************
**                                            **
** CBL3SRC - Call CEE3SRC to set the enclave  **
**                    return code             **
**                                            **
** This is an example of a user-written       **
** condition handler that sets a user         **
** enclave return code and resumes when       **
** a divide-by-zero condition occurs.         **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBL3SRC.

 DATA DIVISION.

 WORKING-STORAGE SECTION.
 01  ENCLAVE-RC             PIC S9(9) BINARY.
 01  FEEDBACK.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
LINKAGE SECTION.
 01  TOKEN                  PIC X(4).
 01  RESULT-CODE            PIC S9(9) BINARY.
     88 RESUME              VALUE +10.
     88 PERCOLATE           VALUE +20.
 01  CURRENT-CONDITION.
     02  Condition-Token-Value.
      COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
```

```
                  04  Cause-Code  PIC S9(4) BINARY.
   03  Case-Sev-Ctl    PIC X.
              03  Facility-ID     PIC XXX.
          02  I-S-Info           PIC S9(9) BINARY.
      01  NEW-CONDITION.
          02  Condition-Token-Value.
          COPY  CEEIGZCT.
              03  Case-1-Condition-ID.
                  04  Severity    PIC S9(4) BINARY.
                  04  Msg-No      PIC S9(4) BINARY.
              03  Case-2-Condition-ID
                        REDEFINES Case-1-Condition-ID.
                  04  Class-Code  PIC S9(4) BINARY.
                  04  Cause-Code  PIC S9(4) BINARY.
              03  Case-Sev-Ctl    PIC X.
              03  Facility-ID     PIC XXX.
          02  I-S-Info           PIC S9(9) BINARY.

      PROCEDURE DIVISION USING CURRENT-CONDITION,
                               TOKEN, RESULT-CODE,
                               NEW-CONDITION.

      HANDLE-CONDITION.
     *************************************************
     ** Check for divide-by-zero condition (CEE349)
     *************************************************
          IF CEE349 of CURRENT-CONDITION  THEN
              MOVE 761 TO ENCLAVE-RC
              CALL "CEE3SRC" USING ENCLAVE-RC,
                                   FEEDBACK
              IF NOT CEE000 of FEEDBACK  THEN
                  DISPLAY "CEE3SRC failed with msg "
                      Msg-No of FEEDBACK UPON CONSOLE
              END-IF
          END-IF.
          SET PERCOLATE TO TRUE

          GOBACK.

      END PROGRAM CBL3SRC.
```

6. Following is an example of a COBOL subroutine that generates a divide-by-zero condition.

```
CBL LIB,QUOTE,C,RENT,OPTIMIZE,NODYNAM
      *Module/File Name: IGZTDIV
      ************************************************
      **                                            **
      **Function       :                            **
      **                                            **
      **     A divide by zero is attempted. This    **
      **  induces the invocation of user condition  **
      **  handler CESETRC registered in program     **
      **  CEGETRC.                                  **
      **               :                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.   CERCDIV.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  TO-DIVIDE        PIC S9(9) BINARY VALUE 1.
       LINKAGE SECTION.
       01  IDIVISOR         PIC S9(9) BINARY.

       PROCEDURE DIVISION USING IDIVISOR.

      ************************************************
      **  divide a constant by IDIVISOR.           **
      ***********************************************
           DIVIDE IDIVISOR INTO TO-DIVIDE.

           GOBACK.
       End program CERCDIV.
```

7. Following is an example in PL/I that sets and retrieves the user enclave return code when a divide-by-zero is generated.

```
*Process lc(101),opt(0),s,map,list,stmt,a(f),ag,macro ;
 /*Module/File Name: IBMDIV                          */
 /****************************************************/
 /**                                                 */
 /** Function: CEE3SRC - Set the enclave return code */
 /**         : CEE3GRC - Get the enclave return code */
 /*                                                  */
 /* 1. A user ZERODIVIDE ON-unit is established by   */
 /*    CESETRC.                                      */
 /* 2. A sub-program, sdivide, is called and causes  */
 /*    a ZERODIVIDE condition to occur.              */
 /* 3. The ON-unit for ZERODIVIDE is entered.        */
 /*    The ON-unit calls CEE3GRC to get the current  */
 /*    enclave return code. It increments the return */
 /*    code by 4444, and sets the enclave return     */
 /*    code to this new value.                       */
 /* 4. On completion, the program prints the enclave */
 /*    return code.                                  */
 /****************************************************/
 CESETRC: Proc Options(Main)       ;

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
                REAL FIXED BINARY(31,0);
    DCL Enclave_RC REAL FIXED BINARY(31,0);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI   /* Instance-Specific Information */

 /****************************************************/
 /* A ZERODIVIDE ON-unit is established              */
 /****************************************************/
    on zerodivide begin;
       call CEE3GRC (Enclave_RC, fc);
       IF  FBCHECK( FC, CEE000)  THEN  DO;
          PUT SKIP LIST( 'Original Enclave RC was '
             || Enclave_RC );
          END;
       ELSE  DO;
          DISPLAY( 'CEE3GRC failed with msg '
             || FC.MsgNo );
          STOP;
          END;
       Enclave_RC = Enclave_RC + 4444;
       call CEE3SRC (Enclave_RC, fc);
       IF  FBCHECK( FC, CEE000)  THEN  DO;
          PUT SKIP LIST( 'New Enclave RC is '
             || Enclave_RC );
          END;
       ELSE  DO;
          DISPLAY( 'CEE3SRC failed with msg '
             || FC.MsgNo );
```

```
         STOP;
         END;
      goto resume;
   end;
/***************************************************/
/* Call sdivide to cause a ZERODIVIDE condition.   */
/***************************************************/
   call sdivide;
resume:
   put skip edit('Enclave return code is ',
         Enclave_RC)  (A, F(10));

/***************************************************/
/* The sdivide routine causes a ZERODIVIDE condition*/
/***************************************************/
sdivide: proc;
   dcl int fixed bin (15,0);
   dcl int_2 fixed bin (15,0) init(5);
   dcl int_3 fixed bin (15,0) init(0);
   int = int_2 / int_3;
end sdivide;

End cesetrc;
```

CEE3GRN—Get name of routine that incurred condition

CEE3GRN gets the name of the most current Language Environment-conforming routine where a condition occurred. If there are nested conditions, the most recently signaled condition is used.

Syntax

```
►►──CEE3GRN──(──name──,──fc──)──────────────────────────────────────►◄
```

name **(output)**
A fixed-length 80-character string (VSTRING), that contains the name of the routine that was executing when the condition was raised. *name* is left-justified within the field and right-padded with blanks. If there are nested conditions, the most recently activated condition is used to determine *name*.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE35S	1	3260	No condition was active when a call to a condition management routine was made.

Usage notes

- z/OS UNIX consideration—In multithread applications, CEE3GRN gets the name of the routine that incurred the condition on the current thread.

Examples

1. Following is an example of CEE3GRN called by C/C++.

```
/*Module/File Name: EDC3GRN   */

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

#ifdef __cplusplus
  extern "C" {
#endif
  void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
  }
#endif

int main(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;

 /* .
    .
    . */
  /* register condition handler */
  token = 99;
  routine.address = (_POINTER)&handler;;
  routine.nesting = NULL;
  CEEHDLR(&routine,&token,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEHDLR failed with message number %d\n",
            fc.tok_msgno);
     exit (2999);
  }

 /*
⋮
 *//* set up any condition sev 2 or higher */
  c_1 = 3;
  c_2 = 99;
  cond_case = 1;
  sev = 3;
  control = 0;
  memcpy(facid,"ZZZ",3);
  isi = 0;
  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&condtok,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  /* signal condition */
  CEESGL(&condtok,&qdata,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEESGL failed with message number %d\n",
            fc.tok_msgno);
     exit (2999);
  }
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _CHAR80 name;
   _FEEDBACK grnfc;

   /* get name of the routine that signal the        */
   /* condition                                      */
   CEE3GRN(name,&grnfc);
  if ( _FBCHECK ( grnfc , CEE000 ) != 0 ) {
     printf("CEESGL failed with message number %d\n",
            grnfc.tok_msgno);
     exit (2999);
  }

   printf("the routine that called this condition");
   printf(" handler is:\n %.80s\n",name);
   *result = 10;
   return;
}
```

2. Following is an example of CEE3GRN called by COBOL.

```
CBL LIB,QUOTE,NOOPT
      *Module/File Name: IGZT3GRN
      *************************************************
      **                                             **
      ** DRV3GRN - Drive sample program for CEE3GRN. **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRV3GRN.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE           PROCEDURE-POINTER.
       01  DENOMINATOR       PIC S9(9) BINARY.
       01  NUMERATOR         PIC S9(9) BINARY.
       01  RATIO             PIC S9(9) BINARY.
       01  TOKEN             PIC S9(9) BINARY VALUE 0.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.

       PROCEDURE DIVISION.

       REGISTER-HANDLER.
      *************************************************
      ** Register handler
      *************************************************
           SET ROUTINE TO ENTRY "CBL3GRN".
```

```
     CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
     IF  NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLR failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
 RAISE-CONDITION.
*************************************************
** Cause a zero-divide condition.
*************************************************
     MOVE 0 TO DENOMINATOR.
     MOVE 1 TO NUMERATOR.
     DIVIDE NUMERATOR BY DENOMINATOR
            GIVING RATIO.

 UNREGISTER-HANDLER.
*************************************************
** UNregister handler
*************************************************
     CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLU failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.

     STOP RUN.
 END PROGRAM DRV3GRN. *************************************************
**                                             **
** CBL3GRN - Call CEE3GRN to get the name of   **
**           the routine that incurred         **
**           the condition.                    **
**                                             **
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBL3GRN.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  RNAME                   PIC X(80).
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 LINKAGE SECTION.
 01  TOKEN                   PIC S9(9) BINARY.
 01  RESULT                  PIC S9(9) BINARY.
     88 RESUME                   VALUE 10.
 01  CURCOND.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
```

```
            02  I-S-Info            PIC S9(9) BINARY.

        01  NEWCOND.
            02  Condition-Token-Value.
            COPY  CEEIGZCT.
                03  Case-1-Condition-ID.
                    04  Severity    PIC S9(4) BINARY.
                    04  Msg-No      PIC S9(4) BINARY.
                03  Case-2-Condition-ID
                          REDEFINES Case-1-Condition-ID.
                    04  Class-Code  PIC S9(4) BINARY.
                    04  Cause-Code  PIC S9(4) BINARY.
                03  Case-Sev-Ctl    PIC X.
                03  Facility-ID     PIC XXX.
            02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION USING CURCOND, TOKEN,
                                 RESULT, NEWCOND.
        PARA-CBL3GRN.
            CALL "CEE3GRN" USING RNAME, FC.
            IF  CEE000 of FC  THEN
                DISPLAY "Name of routine which "
                    "incurred the condition is:  " RNAME
            ELSE
                DISPLAY "CEE3GRN failed with msg "
                    Msg-No of FC UPON CONSOLE
                STOP RUN
            END-IF.

        PARA-HANDLER.
       *************************************************
       ** In user handler - resume execution
       *************************************************
            SET RESUME TO TRUE.

            GOBACK.

        END PROGRAM CBL3GRN.
```

3. Following is an example of CEE3GRN called by PL/I.

```
*PROCESS OPT(0), MACRO;
 /* Module/File Name: IBM3GRN                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3GRN - example of CEE3GRN         **/
 /**                    invoked from PL/I ON-unit  **/
 /**                                                **/
 /****************************************************/

 IBM3GRN:  PROCEDURE  OPTIONS(MAIN);

   %INCLUDE CEEIBMAW;
   %INCLUDE CEEIBMCT;

   DECLARE
      RNAME       CHAR(80),
      01 FC,                      /* Feedback token */
         03 MsgSev    REAL FIXED BINARY(15,0),
         03 MsgNo     REAL FIXED BINARY(15,0),
         03 Flags,
            05 Case      BIT(2),
            05 Severity  BIT(3),
            05 Control   BIT(3),
         03 FacID     CHAR(3),        /* Facility ID */
         03 ISI     /* Instance-Specific Information */
                      REAL FIXED BINARY(31,0),
      divisor     FIXED BINARY(31) INITIAL(0);
      ON  ZERODIVIDE  BEGIN;

      /* Call CEE3GRN to get the name of the routine    */
      /*    that incurred the most recently signalled   */
      /*    condition                                   */

         CALL CEE3GRN ( RNAME, FC );
         IF  FBCHECK( FC, CEE000)  THEN  DO;
            PUT SKIP LIST( 'The most recently signalled '
               || 'condition was incurred by ' || RNAME );
            END;
         ELSE  DO;
            DISPLAY( 'CEE3GRN failed with msg '
               || FC.MsgNo );
            END;

         END /* ON ZeroDivide */;


      divisor = 15 / divisor  /* signals ZERODIVIDE */;

    END IBM3GRN;
```

CEE3GRO—Get offset of condition

The CEE3GRO service returns the offset within a failing routine of the most recent condition. If there are nested conditions, the most recently signaled condition is returned.

Syntax

```
►►──CEE3GRO──(──cond_offset──,──fc──)─────────────────────────────────►◄
```

cond_offset **(output)**
An INT4 data type that, upon completion of this service, contains the offset within a failing routine of the most recent condition.

fc **(output, optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE35S	1	3260	No condition was active when a call to a condition management routine was made.

Examples

1. Following is an example of CEE3GRO called by COBOL.

```
CBL  LIB,QUOTE,NOOPT
      *Module/File Name: IGZT3GRO
      *************************************************
```

```
**                                              **
** DRV3GRO - Register a condition handler       **
**           that calls CEE3GRO to determine    **
**           the offset in the program that     **
**           incurred the condition.            **
**                                              **
**************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DRV3GRO.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ROUTINE      PROCEDURE-POINTER.
 01  DENOMINATOR  PIC S9(9) BINARY.
 01  NUMERATOR    PIC S9(9) BINARY.
 01  RATIO        PIC S9(9) BINARY.
 01  TOKEN        PIC S9(9) BINARY VALUE 0.
 01  FC.

 02  Condition-Token-Value.
     COPY CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
               REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.       REGISTER-HANDLER.
*************************************************
** Register handler
*************************************************
     SET ROUTINE TO ENTRY "CBL3GRO".
     CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
     IF NOT CEE000 of FC THEN
        DISPLAY "CEEHDLR failed with msg "
                Msg-No of FC UPON CONSOLE
        STOP RUN
     END-IF.    RAISE-CONDITION.
*************************************************
** Cause a zero-divide condition
*************************************************
     MOVE 0 TO DENOMINATOR.
     MOVE 1 TO NUMERATOR.
     DIVIDE NUMERATOR BY DENOMINATOR
        GIVING RATIO.

 UNREGISTER-HANDLER.
*************************************************
** Unregister handler
*************************************************
     CALL "CEEHDLU" USING ROUTINE, FC.
     IF NOT CEE000 of FC THEN
        DISPLAY "CEEHDLU failed with msg "
                Msg-No of FC UPON CONSOLE
        STOP RUN
     END-IF.

     STOP RUN.
 END PROGRAM DRV3GRO.

**************************************************
**                                              **
```

```
          ** CBL3GRO - Call CEE3GRO to get the offset   **
          **           in the routine that incurred     **
          **           the condition.                   **
          **                                            **
          ************************************************

           IDENTIFICATION DIVISION.
           PROGRAM-ID.  CBL3GRO.

           DATA DIVISION.
           WORKING-STORAGE SECTION.
           01  ROFFSET  PIC S9(9) BINARY.
           01  FC.
               02  Condition-Token-Value.
               COPY CEEIGZCT.
                   03  Case-1-Condition-ID.
                       04  Severity    PIC S9(4) BINARY.
                       04  Msg-No      PIC S9(4) BINARY.
                   03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                       04  Class-Code  PIC S9(4) BINARY.
                       04  Cause-Code  PIC S9(4) BINARY.
                   03  Case-Sev-Ctl    PIC X.
                   03  Facility-ID     PIC XXX.
               02  I-S-Info            PIC S9(9) BINARY.
     LINKAGE SECTION.
           01  TOKEN        PIC S9(9) BINARY.
           01  RESULT       PIC S9(9) BINARY.
               88  RESUME   VALUE 10.
           01  CURCOND.
               02  Condition-Token-Value.
               COPY CEEIGZCT.
                   03  Case-1-Condition-ID.
                       04  Severity    PIC S9(4) BINARY.
                       04  Msg-No      PIC S9(4) BINARY.
                   03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                       04  Class-Code  PIC S9(4) BINARY.
                       04  Cause-Code  PIC S9(4) BINARY.
                   03  Case-Sev-Ctl    PIC X.
                   03  Facility-ID     PIC XXX.
               02  I-S-Info            PIC S9(9) BINARY.
           01  NEWCOND.
               02  Condition-Token-Value.
               COPY CEEIGZCT.
                   03  Case-1-Condition-ID.
                       04  Severity    PIC S9(4) BINARY.
                       04  Msg-No      PIC S9(4) BINARY.
                   03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                       04  Class-Code  PIC S9(4) BINARY.
                       04  Cause-Code  PIC S9(4) BINARY.
                   03  Case-Sev-Ctl    PIC X.
                   03  Facility-ID     PIC XXX.
               02  I-S-Info            PIC S9(9) BINARY.

           PROCEDURE DIVISION USING CURCOND, TOKEN,
                                    RESULT, NEWCOND.
           PARA-CBL3GRO.
               CALL "CEE3GRO" USING ROFFSET, FC.
               IF CEE000 of FC THEN
                  DISPLAY "Offset in routine which "
                          "incurred the condition is:  "
                          ROFFSET
               ELSE
                  DISPLAY "CEE3GRO failed with msg "
                          Msg-No of FC UPON CONSOLE
```

```
                END-IF.

            PARA-HANDLER.
           *************************************************
           **  In user handler - resume execution
           *************************************************
                SET RESUME TO TRUE.

                GOBACK.
            END PROGRAM CBL3GRO.
```

2. Following is an example of CEE3GRO called by PL/I.

```
*Process lc(101),opt(0),s,map,list,stmt,a(f),ag;
*Process macro;
 DRV3GRO: Proc Options(Main);

   /*Module/File Name: IBM3GRO                      */
   /**************************************************
    **                                              *
    ** DRV3GRO - Register a condition handler that    *
    **           calls CEE3GRO to determine         *
    **           the offset in the routine that     *
    **           incurred the condition.            *
    **                                              *
    *************************************************/

    %include CEEIBMCT;
    %include CEEIBMAW;
    declare 01 FBCODE    feedback;
    declare DENOMINATOR  real fixed binary(31,0);
    declare NUMERATOR    real fixed binary(31,0);
    declare RATIO        real fixed binary(31,0);
    declare PLI3GRO      external entry;
    declare U_PTR        pointer;
    declare 01 U_DATA,
              03 U_CNTL  fixed binary(31,0),
              03 U_TOK   pointer;

    U_PTR = addr(U_DATA);
    U_CNTL = 0;

    /* Set Resume Point               */

    Display('Setting resume point via CEE3SRP');
    Call CEE3SRP(U_TOK,FBCODE);
    Display('After call to CEE3SRP ... Resume point');
    If U_CNTL = 0
      Do;
        Display('First time through...');

        /* Register User Handler  */

        Display('Registering user handler');
        Call CEEHDLR(PLI3GRO, U_PTR, FBCODE);
        If FBCHECK(FBCODE, CEE000) then
          Do;
            /* Cause a zero-divide condition  */
            DENOMINATOR = 0;
            NUMERATOR = 1;
            RATIO = NUMERATOR/DENOMINATOR;
          End;
        Else
          Do;
            Display('CEEHDLR failec with msg');
            Display(MsgNo);
          End;
      End;
    Else
      Display('2nd time...User can do whatever');

    /* Unregister handler  */

    Call CEEHDLU(PLI3GRO, FBCODE);
    If FBCHECK (FBCODE, CEE000) Then
      Display('Main: unregistered PLI3GRO);
    Else
      Do;
        Display('CEEHDLU failed with msg ');
        Display(MsgNo);
      End;
 End DRV3GRO;*Process lc(101),opt(0),s,map,list,stmt,a(f),ag;
*Process macro;
 PLI3GRO: Proc (PTR1,PTR2,PTR3,PTR4);
   /*************************************************
    **                                              *
    ** PLI3GRO - Call CEE3GRO to get the offset in    *
    **           the routine that incurred the       *
    **           condition.                         *
    **                                              *
    *************************************************/

    %include CEEIBMCT;
    %include CEEIBMAW;
    declare (PTR1,PTR2,PTR3,PTR4) pointer;
    declare 01 CURCOND based(PTR1) feedback;
    declare TOKEN      pointer based(PTR2);
    declare RESULT     fixed binary(31,0) based(PTR3);
    declare 01 NEWCOND based(PTR4) feedback;
    declare ROFFSET    real fixed binary(31,0);
    declare 01 FBCODE  feedback;
    declare 01 U_DATA  based(TOKEN),
             03 U_CNTL fixed binary(31,0),
             03 U_TOK  pointer;

         Call CEE3GRO(ROFFSET,FBCODE);
         If fbcheck (fbcode, cee000) Then
           Do;
             Display('Routine offset which incurred');
             Display('the condition is: ');
             Display(ROFFSET);
           End;
         Else
           Do;
             Display('CEE3GRO failed with msg ');
             Display(FBCODE.MsgNo);
           End;

   /*************************************************
    ** In user handler - resume execution           *
    *************************************************/

         RESULT = 10;
         Call CEEMRCE(U_TOK,FBCODE);
         U_CNTL = 1;
         Return;
 End PLI3GRO;
```

CEE3INF—Query enclave information

CEE3INF queries and returns to the calling routine the information about the system/subsystem, the environment information, the member languages, and the version of Language Environment associated with this enclave.

Syntax

```
►►──CEE3INF──(──sys/subsys──,──env-info──,──member-id──,──gpid──,──fc──)──────►◄
```

sys/subsys **(input/output)**

As input, *sys/subsys* is considered to be an address to a fullword. As output, the fullword is a 32-bit map that represents the operating system or subsystem on which the enclave is currently running.

0	Currently executing in the CICS environment
1	Currently executing in a CICS_PIPI environment
2-3	Reserved for other specific CICS environments
4	Currently executing in a TSO environment
5	Currently executing in a Batch environment
6	Currently executing in a z/OS UNIX environment
7-28	Reserved for future use
29	Currently executing on z/VSE®
30	Currently executing on z/OS
31	Reserved.

env-info **(input/output)**

As input, *env-info* is considered to be an address to a fullword. As output, the fullword is a 32-bit map representing the environments that are active in that enclave.

0	Currently executing in a PIPI environment
1	Currently executing in a PIPI-Main environment
2	Currently executing in a PIPI-Sub environment
3	Currently executing in a PIPI-Subdp environment
4	Currently executing in a PICI (Pre-init compatibility interface) environment. For more information, see *z/OS XL C/C++ Programming Guide*.
5	Currently executing in a nested enclave
6	LRR is active in the current enclave
7	Runtime reuse is active in the current environment
8	XPLINK(ON) is in effect in the current enclave
9	POSIX(ON) RTO in effect in the current enclave
10	At least one pthread has been created in this enclave
11	Currently executing on the IPT
12	Multithreaded fork is in effect in the current enclave
13-14	AMODE Init **B'00'** AMODE 24 **B'10'** AMODE 31
15	Currently executing in a PIPI-Maindp environment
16-31	Reserved for future use

member-id **(input/output)**

As input, *member-id* is considered to be an address to a fullword. As output, the fullword is a 32-bit map representing the member languages that are initialized in that enclave.

0	Reserved
1	Reserved
2	Reserved
3	C/C++
4	Reserved
5	COBOL
6	Reserved
7	Fortran
8-9	Reserved
10	PL/I
11	Enterprise PL/I
12-14	Reserved
15	Reserved
16	Reserved
17-23	Reserved for future use
24-31	The current version number of CEE3INF. It is currently set to 0.

gpid **(input/output)**

A fullword integer representing the version of Language Environment that created this thread. This fullword can be interpreted as a four-byte hex number as follows:

```
|PP|VV|RR|MM|
```

PP	product number
VV	version
RR	release
MM	modification

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Usage notes

- z/OS UNIX considerations — CEE3INF is allowed only in the thread.
- If you are not interested in any one of the parameters being passed, a 0 must be placed in the slot of those parameters. This indicates that no information is needed regarding that parameter.
- When the PIPI or nested enclave environment bit is on, a subsystem bit may not be set.

Examples

1. An example of CEE3INF called by C/C++:

```
/*Module/File Name: EDC3INF   */
/*********************************************************************/
/*                                                                   */
/* THIS EXAMPLE CALLS CEE3INF TO GET INFORMATION OF THE CURRENT      */
/* ENCLAVE, LIKE THE SYSTEM/SUBSYSTEM, THE ENVIRONMENT INFORMATION,  */
/* THE MEMBER LANGUAGES USED AND THE VERSION OF LANGUAGE ENVIRONMENT.*/
/*                                                                   */
/*********************************************************************/

#include <leawi.h>
#include <string.h>
#include <ceeedcct.h>

int main(void) {

 _INT4 sys_subsys,env_info,member_id,gpid;
 _FEEDBACK fc;

  /* Calling CEE3INF to get the information */
  CEE3INF(&sys_subsys,&env_info,&member_id,&gpid,fc);

  if ( _FBCHECK(fc,CEE000) != 0 ) {
     printf("CEE3INF failed with message number %d\n", fc.tok_msgno);
  }
  printf("System/Subsystem in hex %08x \n",sys_subsys);
  printf("Enviornment info in hex %08x \n",env_info);
  printf("Member languages in hex %08x \n",member_id);
  printf("GPID information in hex %08x \n",gpid);
  printf("\n");
}
```

2. An example of CEE3INF called by COBOL:

```
CBL LIB,QUOTE
*************************************************
*Module/File Name: IGZTINF
*****************************************************************
**                                                             **
** Function: CEE3INF - Query enclave information               **
**                                                             **
** This example calls CEE3INF to gather data about the current **
** enclave like the system/subsystem, environment information, **
** member languages, and Language Environment version number.  **
**                                                             **
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBL3INF.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
********************************************************
**  Define space for a Language Environment feedback  **
**  code (12 total bytes).                            **
**  Include a copy of CEEIGZCT to get pre-defined     **
**  Language Environment condition tokens.            **
********************************************************
  01  FC.
      02  CONDITION-TOKEN-VALUE.
      COPY  CEEIGZCT.
          03  CASE-1-CONDITION-ID.
             04  SEVERITY     PIC S9(4) BINARY.
             04  MSG-NO       PIC S9(4) BINARY.
          03  CASE-SEV-CTL    PIC X.
          03  FACILITY-ID     PIC XXX.
      02  I-S-INFO            PIC S9(9) BINARY.
********************************************************
**  Define some storage to be used by the Language    **
**  Environment callable service.                     **
********************************************************
 01 SYS-SUBSYS                PIC S9(9) BINARY.
 01 ENV-INFO                  PIC S9(9) BINARY.
 01 MEM-ID                    PIC S9(9) BINARY.
 01 GPID                    PIC S9(9) BINARY.

 PROCEDURE DIVISION.
 MAIN-PROG.
*        *************************************************
*        ** Now call CEE3INF. It returns the results    **
*        ** of the query on the environment and the     **
*        ** feedback code.                              **
*        ** Report error and stop if CEE3INF fails.     **
*        *************************************************
          CALL "CEE3INF" USING SYS-SUBSYS, ENV-INFO, MEM-ID, GPID, FC.
          IF CEE000 OF FC THEN
             DISPLAY "SYS-SUBSYS: " SYS-SUBSYS
             DISPLAY "ENV-INFO  : " ENV-INFO
             DISPLAY "MEM-ID    : " MEM-ID
             DISPLAY "GPID      : " GPID
          ELSE
             DISPLAY "CEE3INF FAILED WITH MSG "
              MSG-NO OF FC UPON CONSOLE
             STOP RUN
          END-IF.
          GOBACK.
```

3. An example of CEE3INF called by PL/I:

```
*PROCESS MACRO;
  /*****************************************************/
  /*Module/File Name: IBM3INF                                         */
  /*******************************************************************/
  /**                                                              **/
  /** Function: CEE3INF - query enclave information                **/
  /**                                                              **/
  /** This example calls CEE3INF to gather data about the current  **/
  /** enclave like the system/subsystem, environment information,  **/
  /** member languages, and Language Environment version number.   **/
  /**                                                              **/
  /*******************************************************************/

  PLI3INF: PROCEDURE OPTIONS (MAIN) REORDER;

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL SYSPRINT  File  Output  Stream;

    DCL SYS_SUB      REAL FIXED BINARY(31,0),
        ENV_INF      REAL FIXED BINARY(31,0),
        MEM_ID       REAL FIXED BINARY(31,0),
        GPID         REAL FIXED BINARY(31,0);

  /*******************************************************/
  /* Declare a Language Environment Feedback token.      */
  /* 12 Total bytes of storage.                          */
  /*******************************************************/
    Declare 01 LE_Feedback_Code,
              03 MsgSev       REAL FIXED BINARY(15,0),
              03 MsgNo        REAL FIXED BINARY(15,0),
              03 Flags,
                 05 Case      BIT(2),
                 05 Severity  BIT(3),
                 05 Control   BIT(3),
              03 FacID        CHAR(3),
              03 ISI          REAL FIXED BINARY(31,0);
  /*******************************************************/
  /* Local declares needed for Messaging Callable Services*/
  /*******************************************************/
    Declare  Msg_dest         REAL FIXED BINARY(31,0);
    Declare  Msg_String       CHAR(255) VARYING;
  Msg_dest = 2;

/*******************************************************/
/* Call CEE3INF to request info about the              */
/* current enclave                                     */
/*******************************************************/

  CALL CEE3INF(SYS_SUB,ENV_INF,MEM_ID,GPID,LE_Feedback_Code);

/*******************************************************/
/* Output the result.                                  */
/*******************************************************/
  Msg_String = 'System-Subsystem       : ' || BIT(SYS_SUB);
  Call CEEMOUT(Msg_String,Msg_Dest,LE_Feedback_Code);

  Msg_String = 'Environment Information: ' || BIT(ENV_INF);
  Call CEEMOUT(Msg_String,Msg_Dest,LE_Feedback_Code);

  Msg_String = 'Member Languages       : ' || BIT(MEM_ID);
  Call CEEMOUT(Msg_String,Msg_Dest,LE_Feedback_Code);
```

CEE3LNG—Set national language

CEE3LNG sets the current national language. You can also use CEE3LNG to query the current national language. Changing the national language changes the languages in which error messages are displayed and printed, the names of the days of the week, and the names of the months. The current national language setting, as well as previous national language settings that have not been discarded, are recorded on the stack on a LIFO (last in, first out) basis. The current national language setting is always on the top of the stack.

There are two methods of changing the default national language setting with CEE3LNG:

- Specify the new national language setting and place it on top of the stack using a *function* value of 1 (SET). This discards the previous default setting.
- Specify the new national language setting and place it on top of the stack using a *function* value of 3 (PUSH). This pushes the previous national language setting down on the stack so that you can later retrieve it by discarding the current setting.

To illustrate the second method, suppose you live in the United States and the code for the United States is specified as a system-level default. If you want to use the French defaults for a certain application, you can use CEE3LNG to PUSH France as the national language setting; then when you want the defaults for the United States, you can POP France from the top of the stack, making the United States the national language setting.

If you specify a *desired_language* that is not available on your system, Language Environment uses the IBM-supplied default ENU (mixed-case U.S. English). If an unknown national language code is specified as a system-level, region-level or CEEUOPT default, a return code of 4 and a warning message is issued.

You can also use the NATLANG runtime option to set the national language.

CEE3LNG affects only the Language Environment NLS and date and time services, not the Language Environment locale callable services or C locale-sensitive functions.

Syntax

```
►►──CEE3LNG──(──function──,──desired_language──,──fc──)──────────────────────►◄
```

function **(input)**
A fullword binary integer that specifies the service to perform. The possible values for *function* are:

1—SET
Establishes the *desired_language* specified in the call to CEE3LNG as the current language. In effect, it replaces the current language on the top of the stack with the *desired_language* that you specify. When setting the national language, the *desired_language* is folded to uppercase. "enu" and "ENU", for example, are considered to be the same national language.

2—QUERY
Identifies the current language on the top of the stack to the calling routine by returning it in the *desired_language* parameter of CEE3LNG. The *desired_language* retained as the result of the QUERY function is in uppercase.

3—PUSH
Pushes the *desired_language* specified in the call to CEE3LNG on to the top of the language stack, making it the current language. Previous languages are retained on the stack on a LIFO basis, making it possible to return to a prior language at a later time.

4—POP
Pops the current language off the stack. The previous language that was pushed on to the stack now becomes the new current language. Upon return to the caller, the *desired_language* parameter contains the discarded language. If the stack contains only one language and would be empty after the call, no action is taken and a feedback code indicating such is returned.

desired_language **(input/output)**
A 3-character fixed-length string. The string is not case-sensitive and is used in the following ways for different *functions*:

If *function* **is:**	**Then** *desired_language***:**
1 or 3	Contains the desired national language identification. In this case, it is an input parameter. Table 23 on page 177 contains a list of national language identifiers. **Note:** Language Environment supports only these national languages: **ENU** Mixed-case U.S. English **UEN** Uppercase U.S. English **JPN** Japanese.
2	Returns the current language on top of the stack. In this case, it is an output parameter.

If *function* is:	Then *desired_language*:
4	Returns the discarded national language. In this case, it is an output parameter.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3BQ	2	3450	Only one language was on the stack when a POP request was made to CEE3LNG. The current language was returned.
CEE3BR	3	3451	The desired language *desired-language* for the PUSH or SET function for CEE3LNG was invalid. No operation was performed.
CEE3BS	3	3452	The function *function* specified for CEE3LNG was not recognized. No operation was performed.

Usage notes

- PL/I MTF consideration—The CEE3LNG callable service is not supported in PL/I multitasking applications.
- PL/I consideration—When running PL/I with POSIX(ON), CEE3LNG is not supported.
- z/OS UNIX consideration
  - CEE3LNG applies to the enclave. Each enclave has a single current national language setting.
  - Language Environment provides z/OS UNIX-compliant locales as part of the C/C++ runtime library. The locales establish the cultural conventions for locale-sensitive functions in this runtime. The CEESETL callable service and other locale callable services depend on the loaded locale. To change the locale, you can use the `setlocale()` C/C++ library function or the CEESETL callable service. Locale values do not affect the settings used by the CEE3LNG callable service. For a complete list of locales, see *z/OS Language Environment Programming Guide*.
  - The CEE3LNG callable services, such as date and time formatting, sorting, and currency symbols are independent of the locale. CEE3LNG affects only Language Environment NLS and date and time services. Mixing CEE3LNG services and locale callable services might produce inconsistent results.

For more information

- See "NATLANG" on page 57 for more information about the NATLANG runtime option.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 411.
- For more information about `setlocale()`, see *z/OS XL C/C++ Programming Guide*.

Examples

1. Following is an example of CEE3LNG called by C/C++.

```
/*Module/File Name: EDC3LNG   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _INT4 function;
  _CHAR3 lang;

  /* Query the current language setting */
  function = 2; /* function 2 is query */
  CEE3LNG(&function,lang,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE3LNG failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  /* if the current language is not mixed-case */
  /* American English set the current language to */
  /* mixed-case American English */
  if (memcmp(lang,"ENU",3) != 0) {
     memcpy(lang,"ENU",3);
     function = 1; /* function 1 is set */
     CEE3LNG(&function,lang,&fc);
     if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEE3LNG failed with message number %d\n",
               fc.tok_msgno);
        exit(2999);
     }
  }
}
```

2. Following is an example of CEE3LNG called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3LNG
      *************************************************
      **                                             **
      ** CBL3LNG - Set national language             **
      **                                             **
      ** In this example, CEE3LNG is called to query **
      ** the current national language setting. If   **
      ** the setting is not mixed-case U.S. English, **
      ** CEE3LNG is called to change the setting.    **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3LNG.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  FUNCTN                 PIC S9(9) BINARY.
       01  LANG                   PIC X(3).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
```

```
                 04  Msg-No      PIC S9(4) BINARY. 03
          Case-2-Condition-ID
                       REDEFINES Case-1-Condition-ID.
                 04  Class-Code  PIC S9(4) BINARY.
                 04  Cause-Code  PIC S9(4) BINARY.
             03  Case-Sev-Ctl    PIC X.
             03  Facility-ID     PIC XXX.
         02  I-S-Info            PIC S9(9) BINARY.

           PROCEDURE DIVISION.
           PARA-3LNGQRY.
          *************************************************
          ** Specify 2 for QUERY function.
          ** Call CEE3LNG to query the current
          **     national language setting
          *************************************************
               MOVE 2 TO FUNCTN.
               CALL "CEE3LNG" USING FUNCTN, LANG, FC.
               IF CEE000 of FC  THEN
                   DISPLAY "Current National Language is: "
                       LANG
               ELSE
                   DISPLAY "CEE3LNG(query) failed with msg "
                       Msg-No of FC UPON CONSOLE
                   STOP RUN
               END-IF.

           PARA-3LNGSET.
          *************************************************
          ** If the current national language is not
          **     mixed-case U.S. English, then call
          **     CEE3LNG with the SET function (1) to
          **     change the national language to mixed-case
          **     U.S. English
          *************************************************
               IF ( LANG IS NOT  = "ENU" )  THEN
                   MOVE 1 TO FUNCTN
                   CALL "CEE3LNG" USING FUNCTN, LANG, FC
                   IF NOT CEE000 of FC  THEN
                       DISPLAY "CEE3LNG(set) failed with msg "
                           Msg-No of FC UPON CONSOLE
                       STOP RUN
                   END-IF
                   DISPLAY "The national language has ",
                       "been changed to mixed-case "
                       "U.S. English (ENU)."
               END-IF.
               GOBACK.
```

3. Following is an example of CEE3LNG called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3LNG                        */
 /***************************************************/
 /**                                                */
 /** Function: CEE3LNG - set national language      */
 /**                                                */
 /** In this example, CEE3LNG is called to query the */
 /** current national language setting.  If the      */
 /** setting is not mixed case American English,     */
 /** CEE3LNMG is called to change the setting to that*/
 /**                                                */
 /***************************************************/
 PLI3LNG: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
```

```
      DCL FUNCTN  REAL FIXED BINARY(31,0);
      DCL LANG    CHARACTER ( 3 );
      DCL 01 FC,                     /* Feedback token */
             03 MsgSev    REAL FIXED BINARY(15,0),
             03 MsgNo     REAL FIXED BINARY(15,0),
             03 Flags,
                05 Case      BIT(2),
                05 Severity  BIT(3),
                05 Control   BIT(3),
             03 FacID     CHAR(3),      /* Facility ID */
             03 ISI   /* Instance-Specific Information */
                          REAL FIXED BINARY(31,0);

      FUNCTN = 2; /* Specify code to query current     */
                  /* national language                 */

      /* Call CEE3LNG with function code 2 to query    */
      /*    national language                          */

      CALL CEE3LNG ( FUNCTN, LANG, FC );
      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST('The current national language '
            || 'is ' || LANG );
         END;
      ELSE  DO;
         DISPLAY('CEE3LNG failed with msg ' || FC.MsgNo);
         STOP;
         END;

      /* If the current language is not mixed-case     */
      /*    American English, set it to mixed-case     */
      /*    American English                           */

      IF  LANG ¬=  'ENU'  THEN DO;
         FUNCTN = 1;
         CALL CEE3LNG ( FUNCTN, 'ENU', FC);
         IF  ¬ FBCHECK( FC, CEE000)  THEN  DO;
            DISPLAY( 'CEE3LNG failed with msg '
               || FC.MsgNo );
            STOP;
            END;  CALL CEE3LNG ( 2, LANG, FC);
         IF  FBCHECK( FC, CEE000)  THEN  DO;
            PUT SKIP LIST('The national language is now '
               || LANG );
            END;
         ELSE  DO;
            DISPLAY( 'CEE3LNG failed with msg '
               || FC.MsgNo );
            STOP;
            END;

         END /* Language is not ENU */;

   END PLI3LNG;
```

4. Table 23 lists the supported national language codes.

Table 23. National language codes

ID	National Language
AFR	Afrikaans
ARA	Arab countries
BGR	Bulgarian
CAT	Catalan
CHT	Traditional Chinese

Table 23. National language codes (continued)

ID	National Language
CHS	Simplified Chinese
CSY	Czech
DAN	Danish
DEU	German
DES	Swiss German
ELL	Greek
ENG	U.K. English
ENU	U.S. English
ESP	Spanish
FIN	Finnish
FRA	French
FRB	Belgian French
FRC	Canadian French
FRS	Swiss French
HEB	Hebrew
HUN	Hungarian
ISL	Icelandic
ITA	Italian
ITS	Swiss Italian
JPN	Japanese
KOR	Korean
NLD	Dutch
NLB	Belgian Dutch
NOR	Norwegian - Bokmal
NON	Norwegian - Nynorsk
PLK	Polish
PTG	Portuguese
PTB	Brazilian Portuguese
RMS	Rhaeto-Romanic
ROM	Romanian
RUS	Russian
SHC	Serbo-Croatian (Cyrillic)
SHL	Serbo-Croatian (Latin)
SKY	Slovakian
SQI	Albanian
SVE	Swedish
THA	Thai
TRK	Turkish
UEN	U.S. uppercase English
URD	Urdu

CEE3MCS—Get default currency symbol

CEE3MCS returns the default currency symbol for the country you specify with *country_code*. For a list of the default settings for a specified country, see Table 32 on page 481.

Syntax

```
►►──CEE3MCS──(──country_code──,──currency_symbol──,──fc──)──────────────►◄
```

`country_code` **(input)**
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 481. *country_code* is not case-sensitive. If no value is specified, the default country code, as set by the COUNTRY runtime option or the CEE3CTY callable service, is used.

`currency_symbol` **(output)**
A 4-character fixed-length string returned to the calling routine. It contains the default currency symbol for the country specified. The currency symbol is left-justified and padded on the right with blanks, if necessary.

`fc` **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3C7	2	3463	The country code *country_code* was not valid for CEE3MCS. The default currency symbol *currency_symbol* was returned.

Usage notes

- If you specify a *country_code* that is not valid, the default currency symbol is X'9F404040'. In the United States, it is shown as a '$'followed by three blanks.
- z/OS UNIX considerations—CEE3MCS applies to the enclave. Every enclave has a single current country setting that has a single currency symbol. Every thread in every enclave has the same default.
- Euro considerations—For countries in the European Union that have adopted the Euro as the legal tender, the currency symbol is represented as a hexadecimal string in the default country settings. The value is the code point for the Euro sign, taken from a typical code page for the given country. Of course, the actual graphical representation depends on the code page in use. See Table 32 on page 481 for a list of country settings.

  Language Environment supports the Euro as the default currency symbol in the following countries: Austria, Belgium, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, the Netherlands, Portugal, and Spain. As more countries pass the economic and monetary union convergence criteria and adopt the Euro as the legal currency, the default currency symbol will replace the national currency symbol with the Euro sign.

For more information

- See "COUNTRY" on page 22 for an explanation of the COUNTRY runtime option.
- See "CEE3CTY—Set default country" on page 133 for an explanation of the CEE3CTY callable service.
- See "CEE3MC2—Get default and international currency symbols" on page 182 for an explanation of the CEE3MC2 callable service.

Examples

1. Following is an example of CEE3MCS called by C/C++.

```
/*Module/File Name: EDC3MCS   */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country;
   _CHAR4 currency;

   /* get the default currency symbol for Canada */
   memcpy(country,"CA",2);
   CEE3MCS(country,currency,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3MCS failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   printf("The default currency symbol for Canada is:"
          " %.2s\n",currency);
}
```

2. Following is an example of CEE3MCS called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3MCS
      *************************************************
      **                                             **
      ** CBL3MCS - Call CEE3MCS to obtain the        **
      **           default currency symbol           **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3MCS.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COUNTRY                PIC X(2).
       01  CURSYM                 PIC X(4).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
```

```
              04  Class-Code  PIC S9(4) BINARY.
              04  Cause-Code  PIC S9(4) BINARY.
          03  Case-Sev-Ctl    PIC X.
          03  Facility-ID     PIC XXX.
      02  I-S-Info            PIC S9(9) BINARY.

  PROCEDURE DIVISION.

  PARA-CBL3MCS.
 ************************************************
 ** Specify country code for the US in the call
 **     to CEE3MCS
 ************************************************
      MOVE "US" TO COUNTRY.
      CALL "CEE3MCS" USING COUNTRY, CURSYM, FC.

 ************************************************
 ** If CEE3MCS runs successfully, display result.
 ************************************************
      IF CEE000 of FC  THEN
          DISPLAY "The default currency symbol "
              "for the " COUNTRY " is: " CURSYM
      ELSE
          DISPLAY "CEE3MCS failed with msg "
              Msg-No of FC UPON CONSOLE
          STOP RUN
      END-IF.

      GOBACK.
```

3. Following is an example of CEE3MCS called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3MCS                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3MCS - Obtain default currency    **/
 /**                     symbol                     **/
 /**                                                **/
 /****************************************************/
 PLI3MCS: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
    DCL COUNTRY  CHARACTER ( 2 );
    DCL CURSYM   CHARACTER ( 4 );
    DCL 01 FC,                      /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);

    COUNTRY = 'US'; /* Specify country code for the  */
                    /* United States                 */

    /* Call CEE3MCS to return currency symbol for    */
    /*    the United States                          */
    CALL CEE3MCS ( COUNTRY, CURSYM, FC );

    /* Print the default currency symbol for the US  */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'The default currency symbol for the '
```

```
             || COUNTRY || ' is "' || CURSYM || '"');
          END;
       ELSE  DO;
          DISPLAY( 'CEE3MCS failed with msg '
             || FC.MsgNo );
          STOP;
          END;

    END PLI3MCS;
```

CEE3MC2—Get default and international currency symbols

CEE3MC2 returns the default currency symbol and international currency symbol for the country you specify with *country_code*. For a list of the default settings for a specified country, see Table 32 on page 481.

Syntax

```
►►—CEE3MC2—(—country_code—,—currency_symbol—,——————————————————————►

►—international_currency_symbol—,—fc—)———————————————————————————►◄
```

`country_code` **(input)**
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 481. *country_code* is not case-sensitive. If no value is specified, the default country code, as set by the COUNTRY runtime option or the CEE3CTY callable service, is used.

`currency_symbol` **(output)**
A 4-character fixed-length string returned to the calling routine. It contains the default currency symbol for the country specified. The currency symbol is left-justified and padded on the right with blanks, if necessary.

`international_currency_symbol` **(output)**
A 3-character alphabetic fixed-length string returned to the calling routine. It contains the international currency symbol for the country specified.

`fc` **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3C2	2	3458	The country code *country_code* was not valid for CEE3MC2. The default currency symbol *currency_symbol* was returned. no international currency symbol was returned.

Usage notes

- If you specify a *country_code* that is not valid, the default currency symbol is X'9F404040'. In the United States, it is shown as a '$' followed by three blanks.
- z/OS UNIX considerations—CEE3MC2 applies to the enclave. Every enclave has a single current country setting that has a single currency symbol. Every thread in every enclave has the same default.
- Euro considerations—For countries in the European Union that have adopted the Euro as the legal tender, the currency symbol is represented as a hexadecimal string in the default country settings. The value is the code point for the Euro sign, taken from a typical code page for the given country. Of course, the actual graphical representation depends on the code page in use. See Table 32 on page 481 for a list of country settings.

  Language Environment supports the Euro as the default currency symbol in the following countries: Austria, Belgium, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, the Netherlands, Portugal, and Spain. As more countries pass the economic and monetary union convergence criteria and adopt the Euro as the legal currency, the default currency symbol will replace the national currency symbol with the Euro sign.

For more information

- See “COUNTRY” on page 22 for an explanation of the COUNTRY runtime option.
- See “CEE3CTY—Set default country” on page 133 for an explanation of the CEE3CTY callable service.
- See “CEE3MCS—Get default currency symbol” on page 179 for an explanation of the CEE3MCS callable service.

Examples

1. Following is an example of CEE3MC2 called by C/C++.

```
/*Module/File Name: EDC3MC2   */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country;
   _CHAR4 currency;
   _CHAR3 international_currency;

   /* get the default currency symbol for Canada */
   memcpy(country,"CA",2);
   CEE3MC2(country,currency,international_currency,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3MC2 failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   printf("The default currency symbol for Canada is:"
          " %.2s\n",currency);
   printf("The international symbol for Canada is:"
          " %.3s\n",international_currency);
}
```

2. Following is an example of CEE3MC2 called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3MC2
      ************************************************
      **                                            **
      ** CBL3MC2 - Call CEE3MC2 to obtain the       **
      **          currency symbols                  **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3MC2.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COUNTRY                 PIC X(2).
       01  CURSYM                  PIC X(2).
       01  INTCURSYM               PIC X(3).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.

       PROCEDURE DIVISION.

       PARA-CBL3MC2.
      ************************************************
      ** Specify country code for the US in the call
      **     to CEE3MC2
      ************************************************
           MOVE "US" TO COUNTRY.
           CALL "CEE3MC2" USING COUNTRY, CURSYM, INTCURSYM, FC.

      ************************************************
      ** If CEE3MC2 runs successfully, display result.
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY "The default currency symbol "
                   "for the " COUNTRY " is: " CURSYM
               DISPLAY "The international currency symbol "
                   "for the " COUNTRY " is: " INTCURSYM
           ELSE
               DISPLAY "CEE3MC2 failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

3. Following is an example of CEE3MC2 called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBM3MC2                          */
 /****************************************************/
 /**                                                 */
 /** Function: CEE3MC2 - Obtain currency symbols     */
 /**                                                 */
 /****************************************************/
 PLI3MC2: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
```

```
   DCL COUNTRY   CHARACTER ( 2 );
   DCL CURSYM    CHARACTER ( 4 );
   DCL INTCURSYM CHARACTER ( 3 );
   DCL 01 FC,                    /* Feedback token */
          03 MsgSev    REAL FIXED BINARY(15,0),
          03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),    /* Facility ID */
          03 ISI                   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);

   COUNTRY = 'US'; /* Specify country code for the  */
                   /* United States                 */

   /* Call CEE3MC2 to return currency symbol for    */
   /*    the United States                          */
   CALL CEE3MC2 ( COUNTRY, CURSYM, INTCURSYM, FC );

   /* Print the default currency symbol for the US  */
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST(
         'The default currency symbol for the '
         || COUNTRY || ' is "' || CURSYM || '"');
      PUT SKIP LIST(
         'The international currency symbol for the '
         || COUNTRY || ' is "' || INTCURSYM || '"');
      END;
   ELSE  DO;
      DISPLAY( 'CEE3MC2 failed with msg '
         || FC.MsgNo );
      STOP;
      END;

END PLI3MC2;
```

CEE3MDS—Get default decimal separator

CEE3MDS returns the default decimal separator for the country specified by *country_code*. For a list of the default settings for a specified country, see Table 32 on page 481.

Syntax

```
►►──CEE3MDS──(──country_code──,──decimal_separator──,──fc──)──────────────►◄
```

country_code **(input)**
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 481. *country_code* is not case-sensitive. If no value is specified, the default country code, as set by the COUNTRY runtime option or the CEE3CTY callable service, is used.

decimal_separator **(output)**
A 2-character fixed-length string containing the default decimal separator for the country specified. The decimal separator is left-justified and padded on the right with a blank.

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3C4	2	3460	The decimal separator '*decimal-separator*' was truncated and was not defined in CEE3MDS.
CEE3C5	2	3461	The country code *country-code* was invalid for CEE3MDS. The default decimal separator '*decimal-separator*' was returned.

Usage notes

- If you specify a *country_code* that is not valid, the default decimal separator is a period (.).
- z/OS UNIX considerations—CEE3MDS applies to the enclave. Every enclave has a single current country setting that has a single decimal separator. Every thread in every enclave has the same default.

For more information

- See "COUNTRY" on page 22 for an explanation of the COUNTRY runtime option.
- See "CEE3CTY—Set default country" on page 133 for an explanation of the CEE3CTY callable service.

Examples

1. Following is an example of CEE3MDS called by C/C++.

```
/*Module/File Name: EDC3MDS   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country,decimal;

   /* get the default decimal separator for Canada */
   memcpy(country,"CA",2);
   CEE3MDS(country,decimal,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3MDS failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   /* print out the default decimal separator */
   printf("The default decimal separator for");
   printf(" Canada is: %.2s\n",decimal);
}
```

2. Following is an example of CEE3MDS called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3MDS
      ************************************************
      **                                            **
      ** CBL3MDS - Call CEE3MDS to get the          **
      **          default decimal separator         **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3MDS.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COUNTRY                 PIC X(2).
       01  DECSEP                  PIC X(2).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.

       PROCEDURE DIVISION.

      ************************************************
      ** Specify the country code for the US in the
      **     call to CEE3MDS.
      ** If call was successful, print result.
      ************************************************
       PARA-CBL3MDS.
           MOVE "US" TO COUNTRY.
           CALL "CEE3MDS" USING COUNTRY, DECSEP, FC.
           IF CEE000 of FC  THEN
               DISPLAY "The default Decimal Separator "
                   "for " COUNTRY " is "" DECSEP """
           ELSE
               DISPLAY "CEE3MDS failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
           GOBACK.
```

3. Following is an example of CEE3MDS called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3MDS                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3MDS - get default decimal        **/
 /**           separator                            **/
 /****************************************************/
 PLI3MDS: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL COUNTRY  CHARACTER ( 2 );
    DCL DECSEP   CHARACTER ( 2 );
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),      /* Facility ID */
          03 ISI   /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);

   COUNTRY = 'US';     /* Specify country code for  */
                       /* the United States         */

   /* Call CEE3MDS to get default decimal           */
   /*    separator for the US                       */
   CALL CEE3MDS ( COUNTRY, DECSEP, FC );

   /*     Print the default decimal separator for   */
   /* the US            */
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST(
         'The default decimal separator for the '
         || COUNTRY || ' is "' || DECSEP || '"' );
      END;
   ELSE  DO;
      DISPLAY( 'CEE3MDS failed with msg '
         || FC.MsgNo );
      STOP;
      END;
END PLI3MDS;
```

CEE3MTS—Get default thousands separator

CEE3MTS returns the default thousands separator for the specified country with *country_code*.

Syntax

```
►►──CEE3MTS──(──country_code──,──thousands_separator──,──fc──)──────────────►◄
```

country_code **(input)**
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 481. *country_code* is not case-sensitive. If no value is specified, the default country code, as set by the COUNTRY runtime option or the CEE3CTY callable service, is used.

thousands_separator **(output)**
A 2-character fixed-length string representing the default thousands separator for the country specified. The thousands separator is left-justified and padded on the right with a blank.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3C8	2	3464	The thousands separator '*thousands-separator*' was truncated and was not defined in CEE3MTS.
CEE3C9	2	3465	The country code *country-code* was invalid for CEE3MTS. The default thousands separator '*thousands-separator*' was returned.

Usage notes

- If you specify a *country_code* that is not valid, the default thousands separator is a comma (,).
- z/OS UNIX considerations—CEE3MTS applies to the enclave. Every enclave has a single current country setting that has a single thousands separator. Every thread in every enclave has the same default.

For more information

- For a list of the default settings for a specified country, see Table 32 on page 481.
- See "COUNTRY" on page 22 for an explanation of the COUNTRY runtime option.
- See "CEE3CTY—Set default country" on page 133 for an explanation of the CEE3CTY callable service.

Examples

1. Following is an example of CEE3MTS called by C/C++.

```
/*Module/File Name: EDC3MTS   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country,thousand;

   /* get the default thousands separator for Canada */
   memcpy(country,"CA",2);
   CEE3MTS(country,thousand,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3MTS failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   /* print out the default thousands separator */
   printf("The default thousands separator for Canada");
   printf(" is:  %.2s\n",thousand);
}
```

2. Following is an example of CEE3MTS called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3MTS
      *************************************************
      **                                             **
      ** CBL3MTS - Call CEE3MTS to obtain the        **
      **           default thousands separator       **
      **                                             **
```

```
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBL3MTS.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  COUNTRY                PIC X(2).
 01  THOUSEP                PIC X(2).
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.

 PARA-CBL3MTS.
*************************************************
** Specify the country code for the US in the
**     call to CEE3MTS.
*************************************************
     MOVE "US" TO COUNTRY.
     CALL "CEE3MTS" USING COUNTRY, THOUSEP, FC.

*************************************************
** If CEE3MTS runs successfully, display result
*************************************************
     IF CEE000 of FC  THEN
         DISPLAY "The default Thousands Separator"
             " for " COUNTRY " is "" THOUSEP """
     ELSE
         DISPLAY "CEE3MDS failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEE3MTS called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3MTS                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3MTS - obtain default thousands   **/
 /** separator                                      **/
 /**                                                **/
 /****************************************************/
 PLI3MTS: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL COUNTRY  CHARACTER ( 2 );
    DCL THOUSEP  CHARACTER ( 2 );
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    COUNTRY = 'US';   /* Specify US as the country   */
                      /* code for the United States  */

    /* Call CEE3MTS to return default thousands      */
    /*    separator for the US                       */
    CALL CEE3MTS ( COUNTRY, THOUSEP, FC );

    /* If CEE3MTS ran successfully print out result  */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'The default thousands separator for the '
          || COUNTRY || ' is "' || THOUSEP || '"' );
       END;
    ELSE  DO;
       DISPLAY( 'CEE3MTS failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLI3MTS;
```

CEE3PRM—Query parameter string

CEE3PRM queries and returns to the calling routine the parameter string specified at invocation of the program. The returned parameter string contains only program arguments. If no program arguments are available, a blank string is returned.

Syntax

```
►►──CEE3PRM──(──char_parm_string──,──fc──)────────────────────────────►◄
```

char_parm_string **(output)**
An 80-byte fixed-length string passed by CEE3PRM. On return from this service, the *char_parm_string* contains the parameter string specified at invocation of the program. If this parameter string is longer than 80 characters, it is truncated. If the parameter string is shorter than 80 characters, the returned string is padded with blanks. If the program argument passed to the service is absent, or is not a character string, *char_parm_string* is blank.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Code	Severity	Message number	Message text
CEE3I1	1	3649	The parameter string returned from CEE3PRM exceeded the maximum length of 80 bytes and was truncated.

Usage notes

- C/C++ consideration—C/C++ users can use the `__osplist` to return a program argument longer than 80 characters.
- z/OS UNIX considerations—CEE3PRM is allowed only in the initial thread.
- You can use the CEE3PR2 callable service to return a program string that is longer than 80 characters.

For more information

- For more information about the CEE3PR2 callable service, see "CEE3PR2—Query parameter string long" on page 194

Examples

1. Following is an example of CEE3PRM called by C/C++.

```
/*Module/File Name: EDC3PRM   */

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>

int main() {

   _CHAR80 parm;
   _FEEDBACK fc;

   CEE3PRM(parm,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3PRM failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   printf("%.80s\n",parm);
}
```

2. Following is an example of CEE3PRM called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3PRM
      ************************************************
      **                                            **
      ** CBL3PRM - Call CEE3PRM to query the        **
      **           parameter string                 **
      **                                            **
      ** In this example, a call is made to         **
      ** CEE3PRM to return the parameter string     **
      ** that was specified at invocation of the    **
      ** program. The string is then displayed.     **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3PRM.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  PARMSTR                 PIC X(80).
```

```
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.

 PARA-CBL3PRM.
     CALL "CEE3PRM" USING PARMSTR, FC.
     IF CEE000 THEN
         DISPLAY "Program arguments specified: "
          "'" PARMSTR "'"
     ELSE
         DISPLAY "CEE3PRM failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEE3PRM called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3PRM                       */
 /**************************************************/
 /**                                              **/
 /** Function: CEE3PRM - Query Parameter String   **/
 /**                                              **/
 /**************************************************/
 PLI3PRM: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL PARMSTR CHAR(80);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

   /* Call CEE3PRM to return the program arguments   */
   /* specified at invocation of the program         */
    CALL CEE3PRM ( PARMSTR, FC );

   /* There are no non-zero feedback codes to        */
   /* check, so print result                         */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'These program arguments were specified:  "'
          || PARMSTR || '"');
       END;
    ELSE  DO;
       DISPLAY( 'CEE3PRM failed with msg '
                  || FC.MsgNo );
              STOP;
              END;

        END PLI3PRM;
```

CEE3PR2—Query parameter string long

CEE3PR2 queries and returns to the calling routine the parameter string and its associated length specified at invocation of the program. The returned parameter string contains only program arguments. If no program arguments are available, a blank string is returned.

Syntax

```
►►──CEE3PR2──(──int_parm_length──,──char_parm_string──,──fc──)──────────────►◄
```

int_parm_length **(input/output)**
A fullword integer that indicates the length of the parameter string.

- For input: If the value is less than or equal to 0, *int_parm_length* is a request for the length of the parameter string passed during program invocation. If the value is greater than 0, it is the actual size of the *char_parm_string* buffer passed by the caller.
- For output: CEE3PR2 returns the actual length of the parameter string passed during program invocation for any case, regardless of the input value in the *int_parm_length* field.

char_parm_string **(input/output)**

- For input: *char_parm_string* contains the address of a piece of storage that is obtained by the caller.

  Note: This storage is used as a varying string. It must be 2 bytes longer than the length of the parameter string passed during program invocation.
- For output: The storage pointed to by this address contains a varying string that contains the parameter string passed during program invocation. The parameter string is truncated when the actual length of the parameter string passed during invocation exceeds *int_parm_length* minus 2.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3K3	1	3715	The parameter string returned from CEE3PR2 was truncated due to insufficient storage space for the string provided by the caller.

Usage notes

- C/C++ consideration—C/C++ users can use the `__osplist` return a program argument longer than 80 characters.
- z/OS UNIX consideration—CEE3PR2 is allowed only in the initial thread.
- CICS consideration—CEE3PR2 always returns a blank string.

For more information

- For more information about the CEE3PRM callable service, see "CEE3PRM—Query parameter string" on page 191.

Examples

1. Following an example of CEE3PR2 called by C/C++.

```
/*Module/File Name: EDC3PR2    */
/*********************************************************************/
/*                                                                   */
/* THIS EXAMPLE CALLS CEE3PR2 TO RETRIEVE THE PARAMETER STRING THAT  */
/* WAS SPECIFIED AT THE INVOCATION OF THE PROGRAM AND ITS ASSOCIATED */
/* LENGTH. BOTH THESE VALUES ARE THEN PRINTED.                       */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>

struct mystring{
    unsigned short int length;
    char           string[298];
};
typedef struct mystring mystring;

int main() {

    int len;

    mystring parm;
    _FEEDBACK fc;

    /*Setting up the user defined space to store the retrieved string*/
    memset(&parm,0x00,sizeof(parm));
    len = sizeof(parm);

    /*Calling CEE3PR2 to retrieve the parameter string*/
    CEE3PR2(&len,&parm,&fc);

    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
       printf("CEE3PR2 failed with message number %d\n", fc.tok_msgno);
    }

    printf("Length field of the string : %d \n",len);
    printf("Parameter String is: %s\n",parm.string);

    return 0;
}
```

2. Following is an example of CEE3PR2 called by COBOL.

```
CBL LIB,QUOTE
      *************************************************
      *Module/File Name: IGZTPR2
      ************************************************
      **                                            **
** CBL3PR2 - Call CEE3PR2 to query the        **
** parameter string                           **
**                                            **
** In this example, a call is made to         **
** CEE3PR2 to return the parameter string     **
** that was specified at invocation of the    **
** program. The string is then displayed.     **
**                                            **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBL3PR2.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 PARMLEN                 PIC S9(9) BINARY.
 01 PARMSTR.
    02 STR1-LENGTH PIC S9(4) BINARY.
    02 STR1-STRING.
       03 STR1-CHAR PIC X
                    OCCURS 0 TO 256 TIMES
                    DEPENDING ON STR1-LENGTH.
 01 FC.
    02 CONDITION-TOKEN-VALUE.
    COPY CEEIGZCT.
        03 CASE-1-CONDITION-ID.
          04 SEVERITY      PIC S9(4) BINARY.
          04 MSG-NO        PIC S9(4) BINARY.
        03 CASE-SEV-CTL    PIC X.
        03 FACILITY-ID     PIC XXX.
     02 I-S-INFO           PIC S9(9) BINARY.

 PROCEDURE DIVISION.

 PARA-CBL3PR2.
     MOVE 258 TO PARMLEN
     CALL "CEE3PR2" USING PARMLEN, PARMSTR, FC.
     IF CEE000 THEN
         DISPLAY "Program arguments specified: '"
             STR1-STRING "'"
     ELSE
         DISPLAY "CEE3PR2 FAILED WITH MSG "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
     GOBACK.
```

3. Following is an example of CEE3PR2 called by PL/I.

```
*PROCESS MACRO;
  /***************************************************/
  /*Module/File Name: IBM3PR2                                       */
  /*****************************************************************/
  /**                                                             **/
  /** Function: CEE3PR2 - Query Parameter String                  **/
  /**                                                             **/
  /** This example calls CEE3PR2 to retrieve the arguments passed **/
  /** during the invocation of the program.                       **/
  /**                                                             **/
  /*****************************************************************/

  PLI3PR2: PROCEDURE OPTIONS (MAIN) REORDER;

    %INCLUDE CEEIBMAW;
    %INCLUDE CEEIBMCT;

    DCL SYSPRINT File Output Stream;

    DCL Parm_Len    FIXED BINARY(31,0),
      Parm_Str    CHAR(255) VARYING;

/*******************************************************/
/* Declare a Language Environment Feedback token.      */
/* 12 Total bytes of storage.                          */
/*******************************************************/
  Declare 01 LE_Feedback_Code,
            03 MsgSev       REAL FIXED BINARY(15,0),
            03 MsgNo        REAL FIXED BINARY(15,0),
            03 Flags,
               05 Case      BIT(2),
               05 Severity  BIT(3),
               05 Control   BIT(3),
            03 FacID        CHAR(3),
            03 ISI          REAL FIXED BINARY(31,0);

/*******************************************************/
/* Local declares needed for Messaging Callable Services*/
/*******************************************************/
  Declare  Msg_String       CHAR(255) VARYING;
  Declare  Output           PICTURE '99999';

  Msg_dest = 2;

  Parm_Len = 255; /*******************************************************/
/* Call CEE3PR2 to get the parameter string            */
/*******************************************************/

  CALL CEE3PR2(Parm_Len,Parm_Str,LE_Feedback_Code);

  If ^ FBCHECK(LE_Feedback_Code, CEE000) Then Do;
    Msg_String = 'Truncation occured';
    Call CEEMOUT(Msg_String,Msg_Dest,LE_Feedback_Code);
  End;
  Else Do;
    /***************************************************/
    /* Output the result.                              */
    /***************************************************/
    MsgString = 'The returned parameter string is: ';
```

CEE3RPH—Set report heading

CEE3RPH sets the heading displayed at the top of the storage or options report generated when you specify the RPTSTG(ON) or RPTOPTS(ON) runtime options. For examples of these reports, see *z/OS Language Environment Debugging Guide*.

Syntax

```
►►—CEE3RPH—(—report_heading—,—fc—)————————————————————————————►◄
```

report_heading **(input)**
An 80-character fixed-length string. *report_heading* sets the identifying character string displayed at the top of the storage or options report. Language Environment uses only the first 79 bytes of *report_heading*; the last byte is ignored. *report_heading* can contain DBCS characters surrounded by X'0E' (shift-out) and X'0F' (shift-in).

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result

of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3JK	0	3700	The storage and options report heading replaced a previous heading.

Usage notes

- PL/I considerations—CEE3RPH is designed to provide an equivalent function to the special PL/I variable PLIXHD.
- z/OS UNIX considerations—CEE3RPH applies to the enclave.

For more information

- See “RPTSTG” on page 71 for more information about the RPTSTG runtime option.
- See “RPTOPTS” on page 69 for more information about the RPTOPTS runtime option.
- See *PL/I for MVS & VM Programming Guide* for further information about PLIXHD.

Examples

1. Following is an example of CEE3RPH called by C/C++.

```
/*Module/File Name: EDC3RPH   */

#pragma runopts(RPTOPTS(ON))
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _CHAR80   heading;

   /* initialize heading to blanks and then set the */
   /* heading */
   memset(heading,' ',80);
   memcpy(heading,"User Defined Report Heading",27);

   /* set the report heading...do not need to check */
   /* feedback token because all return codes are */
   /* successful */
   CEE3RPH(heading,NULL);
  /* .
     .
     . */
}
```

2. Following is an example of CEE3RPH called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3RPH
      ************************************************
      **                                            **
      ** CBL3RPH - Call CEE3RPH to set report heading**
      **                                            **
      ** In this example, a call is made to CEE3RPH  **
** to set the report heading that appears at   **
** the top of each page of the options report  **
** (generated by RPTOPTS) and storage report   **
** (generated by RPTSTG).                      **
**                                             **
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBL3RPH.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  RPTHEAD                 PIC X(80).
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.

*************************************************
** Specify user-defined report heading via CEE3RPH
*************************************************
 PARA-CBL3RPH.
     MOVE "My options and storage reports heading"
         TO RPTHEAD.
     CALL "CEE3RPH" USING RPTHEAD, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEE3RPH failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEE3RPH called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3RPH                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3RPH - set report heading         **/
 /**                                                **/
 /****************************************************/
 PLI3RPH: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL RPTHEAD  CHAR(80);
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
```

```
          /* Define report heading                           */
          RPTHEAD = 'My storage report heading';

          /* Set report heading in call to CEE3RPH          */
          CALL CEE3RPH ( RPTHEAD, FC );

          IF  FBCHECK( FC, CEE000)  THEN  DO;
             PUT SKIP LIST( 'Report heading now set to "'
                || RPTHEAD || '"' );
             END;
          ELSE  DO;
             DISPLAY( 'CEE3RPH failed with msg '
                || FC.MsgNo );
             STOP;
             END;

       END PLI3RPH;
```

CEE3SPM—Query and modify Language Environment hardware condition enablement

CEE3SPM queries and modifies the enablement of Language Environment hardware conditions. You can use the CEE3SPM service to:

- Alter the settings of the Language Environment conditions, using an *action* value of 1 (SET), to those specified by the caller.
- Query the current settings of the Language Environment conditions and return the settings to the caller.
- Push the current settings of the Language Environment conditions on to the condition stack using an *action* value of 3 (PUSH). This pushes the current setting down on the stack for later retrieval, and, in effect, places a copy of the current setting on top of the stack. It does not alter the current condition settings.
- Pop the pushed settings of the Language Environment conditions from the condition stack using an *action* value of 4 (POP). This reinstates the previous condition settings as the current condition settings.
- Push the current settings of the Language Environment conditions on to the Language Environment-managed condition stack and alter the settings of the Language Environment conditions to those supplied by the caller.

The enabled or disabled Language Environment conditions are:

fixed-overflow
When enabled, raises the fixed-overflow condition when an overflow occurs during signed binary arithmetic or signed left-shift operations.

decimal-overflow
When enabled, raises the decimal-overflow condition when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the results.

underflow
When enabled, raises the underflow condition when the result characteristic of a floating-point operation is less than zero and the result fraction is not zero. For an extended-format floating-point result, the condition is raised only when the high-order characteristic underflows.

significance
: When enabled, raises the significance condition when the resulting fraction in floating-point addition or subtraction is zero.

When you use the CEE3SPM callable service, maintenance of the condition stack is required. For example, one user-written condition handler can disable a hardware condition while another enables it. Therefore, do not assume that the program mask is at a given setting. The program mask is set differently based on different HLL requirements. You can find out what the current setting is by using the QUERY function of CEE3SPM.

Language Environment initialization sets conditions based on the languages in the initial load module. Each language present adds to the conditions that are enabled.

Some S/370 hardware interrupt codes and their matching Language Environment feedback codes appear in Table 24 on page 203.

Syntax

```
►►──CEE3SPM──(──action──,──cond_string──,──fc──)──────────────────────────►◄
```

action **(input)**
The action to be performed. *action* is specified as a fullword binary signed integer corresponding to one of the numbers in the following list:

1—SET
Alters the settings of the Language Environment conditions to those specified in the *cond_string* parameter.

2—QUERY
Queries the current settings of the Language Environment conditions and return the settings in the *cond_string* parameter.

3—PUSH
Pushes the current settings of the Language Environment conditions on to the Language Environment-managed condition stack.

4—POP
Pops the pushed settings of the Language Environment conditions from the condition stack, discarding the current settings, and reinstating the previous condition settings as the current condition settings.

5—PUSH, SET
Pushes the current settings of the Language Environment conditions on to the Language Environment-managed condition stack and alters the settings of the Language Environment conditions to those supplied by the caller in the *cond_string* parameter.

cond_string **(input/output)**
A fixed-length string of 80 bytes containing a sequence of identifiers representing the requested settings for the Language Environment conditions that can be enabled and disabled. A list of conditions enabled and disabled and their associated identifiers is given below:

Condition
Identifier

fixed-overflow
F (NOF for disablement)

decimal-overflow
D (NOD for disablement)

underflow
U (NOU for disablement)

significance
S (NOS for disablement)

An identifier with the 'NO' prefix is used to disable the condition it represents. An identifier without the 'NO' prefix is used to enable the condition that it represents. For example, the token 'F' is used to enable the fixed-overflow condition. The identifier 'NOF' is used to disable the fixed-overflow condition. The rightmost option takes effect in the event of a conflict. Identifiers are separated by blanks or commas.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE36V	4	3295	The condition string from CEE3SPM did not contain all of the settings, because the returned string was truncated.
CEE370	4	3296	Some of the data in the condition string from CEE3SPM could not be recognized.
CEE371	4	3297	The service completed successfully for recognized conditions, unsuccessfully for unrecognized (unsupported) conditions.
CEE372	4	3298	A call to CEE3SPM attempted to PUSH settings onto a full stack.
CEE373	4	3299	A call to CEE3SPM attempted to POP settings off an empty stack.
CEE374	4	3300	The action parameter in CEE3SPM was not one of the digits 1 to 5.

Usage notes

- PL/I MTF consideration—In PL/I MTF applications, CEE3SPM affects only the calling task.
- C/C++ consideration—C/C++ ignores the fixed-overflow, decimal-overflow, underflow, and significance interrupts, no matter what you specify in CEE3SPM.
- COBOL consideration—COBOL ignores the fixed-overflow and decimal-overflow interrupts, no matter what you specify in CEE3SPM.
- z/OS UNIX consideration—In multithread applications, CEE3SPM affects only the calling thread.
- You cannot use CEE3SPM to enable the fixed-overflow, decimal-overflow, underflow or significance interrupts. You can, however, query the settings of CEE3SPM.

Table 24. S/370 interrupt code descriptions

S/370 interrupt code	Description	Maskable	Symbolic feedback code	Message number	Severity
0001	Operation exception	No	CEE341	3201	3
0002	Privileged operation exception	No	CEE342	3202	3
0003	Execute exception	No	CEE343	3203	3
0004	Protection exception	No	CEE344	3204	3
0005	Addressing exception	No	CEE345	3205	3
0006	Specification exception	No	CEE346	3206	3
0007	Data exception	No	CEE347	3207	3
0008	Fixed-point overflow exception	Yes	CEE348	3208	3
0009	Fixed-point divide exception	No	CEE349	3209	3
000A	Decimal-overflow exception	Yes	CEE34A	3210	3
000B	Decimal divide exception	No	CEE34B	3211	3
000C	Exponent-overflow exception	No	CEE34C	3212	3
000D	Exponent-underflow exception	Yes	CEE34D	3213	3
000E	Significance exception	Yes	CEE34E	3214	3
nn0F	Floating-point divide exception	No	CEE34F	3215	3

Examples

1. Following is an example of CEE3SPM called by C/C++.

```
/*Module/File Name: EDC3SPM   */

/****************************************************/
/* This example queries the enablement of LE/370     */
/* hardware conditions.                              */
/****************************************************/
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _INT4 action;
  _CHAR80 cond_string;
  char *cond;

  /* query the current settings */
  action = 2;
  CEE3SPM(&action,cond_string,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE3SPM query failed with message %\n",
            fc.tok_msgno);
     exit(2999);
   }
}
```

2. Following is an example of CEE3SPM called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3SPM
      *************************************************
```

```
**                                              **
** CBL3SPM - Call CEE3SPM to query and modify  **
**           Lang. Environ. hardware condition  **
**           enablement                          **
** In this example, a call is made to CEE3SPM  **
** to check the setting of the program mask.   **
** See the parameter list of CEE3SPM to        **
** interpret what is returned as CONDSTR in    **
** this example.                                **
**                                              **
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBL3SPM.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ACTION                  PIC S9(9) BINARY.
 01  CONDSTR                 PIC X(80).
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.

*************************************************
** Specify 2 for the QUERY function.
** Pass ACTION in the call to CEE3SPM to return
**     the condition string DISPLAY results.
*************************************************
 PARA-CBL3SPM.
     MOVE 2 TO ACTION.
     CALL "CEE3SPM" USING ACTION, CONDSTR, FC.
     IF  CEE000 of FC  THEN
         DISPLAY "The current setting of the ",
             "program mask is: " CONDSTR
     ELSE
         DISPLAY "CEE3SPM failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEE3SPM called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3SPM                        */
 /***************************************************/
 /**                                                 */
 /** Function: CEE3SPM - Query and Modify LE/370     */
 /**                  Hardware Condition Enablement */
 /**                                                 */
 /** This example calls CEE3SPM to query the current */
 /** setting of the program mask. See the parameter  */
 /** list of CEE3SPM to interpret what is returned   */
 /** as CONDSTR in this example.                     */
 /**                                                 */
 /***************************************************/
```

```
PLI3SPM: PROC OPTIONS(MAIN);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL ACTION   REAL FIXED BINARY(31,0);
   DCL CONDSTR  CHAR(80);
   DCL 01 FC,                     /* Feedback token */
          03 MsgSev    REAL FIXED BINARY(15,0),
          03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),      /* Facility ID */
          03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);

   /* Call CEE3SPM to query the current setting of  */
   /*    the program mask                           */

   ACTION = 2; /* Specify action code 2 to query    */
               /* the program mask                  */
   CALL CEE3SPM ( ACTION, CONDSTR, FC );
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST(
        'The initial setting of the program mask is: '
        || CONDSTR );
      END;
   ELSE  DO;
      DISPLAY('CEE3SPM failed with msg ' || FC.MsgNo);
      STOP;
      END;

/* Call CEE3SPM to enable specification exceptions. */

/* Specify action code 1 to SET the program mask.   */
   ACTION = 1;

/* Specify a program mask that allows specification */
/* exceptions (all others are unchanged)            */
   CONDSTR = 'S'; CALL CEE3SPM ( ACTION, CONDSTR, FC );
   IF  ¬ FBCHECK( FC, CEE000)  THEN  DO;
      DISPLAY('CEE3SPM failed with msg ' || FC.MsgNo);
      STOP;
      END;

   CALL CEE3SPM (2, CONDSTR, FC); /* Query settings */
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST(
         'The new setting of the program mask is: '
         || CONDSTR );
      END;
   ELSE  DO;
      DISPLAY('CEE3SPM failed with msg ' || FC.MsgNo);
      STOP;
      END;

END PLI3SPM;  CALL CEE3SPM ( ACTION, CONDSTR, FC );
   IF  ¬ FBCHECK( FC, CEE000)  THEN  DO;
      DISPLAY('CEE3SPM failed with msg ' || FC.MsgNo);
      STOP;
      END;

   CALL CEE3SPM (2, CONDSTR, FC); /* Query settings */
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST(
                'The new setting of the program mask is: '
                || CONDSTR );
             END;
          ELSE  DO;
             DISPLAY('CEE3SPM failed with msg ' || FC.MsgNo);
             STOP;
             END;

       END PLI3SPM;
```

CEE3SRC—Set the enclave return code

CEE3SRC sets the user enclave return code. The value set is used in the calculation of the final enclave return code at enclave termination.

Syntax

```
►►──CEE3SRC──(──return_code──,──fc──)──────────────────────────────►◄
```

return_code **(input)**
An INT4 data type. The enclave return code to be set should be <= 999,999 and >= 0 to be in the Language Environment-preferred range. (The initial value is 0.)

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0HA	1	0554	A value outside the preferred range of 0 through 999,999 was supplied. However, the value was still used as the enclave return code.

Usage notes

- z/OS UNIX consideration—CEE3SRC is not supported in multithread applications.
- PL/I MTF consideration—For PL/I multitasking applications, the user return code value set during invocation of CEE3SRC affects the current task only. If a PL/I program causes an enclave to terminate, the value set by CEE3SRC at the associated task level is reflected in the final return code at enclave termination.
- PL/I consideration—When running PL/I with POSIX(ON), CEE3SRC is not supported.

For more information

- See *z/OS Language Environment Programming Guide*, for more information about the CEE3SRC callable service.

Examples

CEE3SRC is used with CEE3GRC; see the examples for CEE3GRC show in "Examples" on page 150.

CEE3SRP—Set resume point

The CEE3SRP service sets the resume point at the next instruction in the calling routine. CEE3SRP works only in conjunction with the CEEMRCE service.

Syntax

```
►►──CEE3SRP──(──resume_token──,──fc──)─────────────────────────────────►◄
```

resume_token **(output)**
An INT4 data type that, upon completion of this service, contains a token that represents a machine state block, which Language Environment allocates from heap storage. Language Environment automatically frees the heap storage for the machine state block when the routine associated with the stack frame to which it points returns to its caller.

fc **(output / optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE390	3	3360	The stack frame was not found on the call chain.

Usage notes

- An example for Service Label under COBOL follows:

```
    SET-RECOVERY-POINT-PARAGRAPH.
       CALL 'CEE3SRP' USING RECOVERY-POINT FC.
       SERVICE LABEL.
    ERROR-PARAGRAPH.
* code to do post-error processing
```

- Use the CEE3SRP service only with the CEEMRCE service from within a user condition handler. The token returned by this service is used as input to the CEEMRCE service.
- Language Environment automatically frees the heap storage for the machine state block when the routine associated with the stack frame to which it points returns to its caller. Attempts to use the machine state block after it is freed result in unpredictable behavior.
- COBOL considerations

– A Service Label compiler directive must be specified immediately after the call to CEE3SRP.
– When a resume occurs and control resumes to the next instruction following the call to CEE3SRP, the COBOL RETURN-CODE special register contains an unpredictable value.

Examples

For examples of how to use CEE3SRP in combination with CEEMRCE and CEEHDLR, see "CEEMRCE—Move resume cursor explicit" on page 358.

CEE3USR—Set or query user area fields

CEE3USR sets or queries one of two 4-byte fields known as the user area fields. The user area fields are associated with an enclave and are maintained on an enclave basis. A user area field can be used by vendor or application programs to store a pointer to a global data area or to keep a recursion counter.

Be careful not to confuse the Language Environment user area fields with the PL/I user area. The PL/I user area is a 4-byte field in the PL/I TCA and can be accessed only through assembler language. The PL/I user area continues to be supported for compatibility.

Language Environment initializes both user area fields to X'00000000' during enclave initialization.

Syntax

```
►►──CEE3USR──(──function_code──,──field_number──,──field_value──,──fc──)──────►◄
```

function_code **(input)**
A fullword binary integer representing the function performed:

1—SET
User area field according to the value specified in *field_value*.

2—QUERY
User area field; return current value in *field_value*.

field_number **(input)**
A fullword binary integer indicating the field to set or query. *field_number* must be specified as either 1 or 2.

field_value **(input/output)**
A fullword binary integer.

If *function_code* is specified as 1 (meaning SET user area field), *field_value* contains the value to be copied to the user area field.

If *function_code* is specified as 2 (meaning QUERY user area field), the value in the user area field is copied to *field_value*.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result

of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3PS	3	3900	The function code passed to CEE3USR was not 1 or 2.
CEE3PT	3	3901	The field number passed to CEE3USR was not 1 or 2.

Usage notes

- z/OS UNIX consideration—CEE3USR applies to the enclave.

Examples

1. Following is an example of CEE3USR called by C/C++.

```
/*Module/File Name: EDC3USR   */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>

typedef struct {
   int value1,value2,value3;
   char slot1_80¨;
} info_struct;

int main (void) {

   _INT4 function_code, field_number, field_value;
   _FEEDBACK fc;
   info_struct *info;

   info = (info_struct *)malloc(sizeof(info_struct));
  /* .
     .
     . */
   /* Set User field 1 to point to info_struct */
   function_code = 1;
   field_number = 1;
   field_value = (int)info;

   CEE3USR(&function_code,&field_number,&field_value,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3USR failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
  /* .
     .
     . */
   /* get the value of field 2 */
   function_code = 2;
   field_number = 1;

   CEE3USR(&function_code,&field_number,&field_value,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3USR failed with message number %d\n",
            fc.tok_msgno);
         exit(2999);
      }
     /* .
        .
        . */
   }
```

2. Following is an example of CEE3USR called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZT3USR
      *************************************************
      **                                             **
      ** CBL3USR - Call CEE3USR to set or query user **
      **           area fields                       **
      **                                             **
      ** In this example, CEE3USR is called twice:   **
      ** once to set the value of a user area, and   **
      ** once to query it.                           **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL3USR.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  FUNCODE                 PIC S9(9) BINARY.
       01  FIELDNO                 PIC S9(9) BINARY.
       01  INVALUE                 PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.

      *************************************************
      ** Specify 1 for SET function.
      ** Specify field number 1 to set the value field
      **     number 1.
      ** Specify 23 to make the value of field number 1
      **     equal to 23.
      *************************************************
       PARA-3USRSET.
           MOVE 1  TO FUNCODE.
           MOVE 1  TO FIELDNO.
           MOVE 23 TO INVALUE.
           CALL "CEE3USR" USING FUNCODE, FIELDNO,
                                INVALUE, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY "CEE3USR failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
      *************************************************
      ** Specify 2 for QUERY function.
      ** Specify field number 1 to query the value
      **     of field number 1.
      *************************************************
```

```
    PARA-3USRQRY.
        MOVE 2 TO FUNCODE.
        MOVE 1 TO FIELDNO.
        CALL "CEE3USR" USING FUNCODE, FIELDNO,
                             INVALUE, FC.
        IF CEE000 of FC  THEN
            DISPLAY "User Area field " FIELDNO
                " is: " INVALUE
        ELSE
            DISPLAY "CEE3USR failed with msg "
                Msg-No of FC UPON CONSOLE
            STOP RUN
        END-IF.

        GOBACK.
```

3. Following is an example of CEE3USR called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBM3USR                         */
 /****************************************************/
 /**                                                **/
 /** Function: CEE3USR - set/query user area fields **/
 /**                                                **/
 /** In this example, CEE3USR is called twice: once **/
 /** to set the value of a user area, and once to   **/
 /** query it.                                      **/
 /****************************************************/
 PLI3USR: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL FUNCODE   REAL FIXED BINARY(31,0);
    DCL FIELDNO   REAL FIXED BINARY(31,0);
    DCL OUTVALUE  REAL FIXED BINARY(31,0);
    DCL INVALUE   REAL FIXED BINARY(31,0);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    FUNCODE = 1; /* Specify 1 for the set function   */
    FIELDNO = 1; /* Specify field 1 of two           */
    INVALUE = 5; /* Value to put in field 1          */
    /* Call CEE3USR to set user field 1 to 5         */
    CALL CEE3USR ( FUNCODE, FIELDNO, INVALUE, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'LE/370 User field ' || FIELDNO
          || ' has been set to ' || INVALUE );
       END;
    ELSE  DO;
       DISPLAY( 'CEE3USR failed with msg '
          || FC.MsgNo );
       STOP;
       END;

    /* Call CEE3USR to query the value of field 1    */

    FUNCODE = 2; /* Specify 2 for query function     */
    FIELDNO = 1; /* Specify field 1 of two           */
```

```
        CALL CEE3USR ( FUNCODE, FIELDNO, OUTVALUE, FC );
        IF  FBCHECK( FC, CEE000)  THEN  DO;
           PUT SKIP LIST( 'LE/370 User field ' || FIELDNO
              || ' is currently set to ' || OUTVALUE );
           END;
        ELSE  DO;
           DISPLAY( 'CEE3USR failed with msg '
              || FC.MsgNo );
           STOP;
           END;

     END PLI3USR;
```

CEECBLDY—Convert date to COBOL Integer format

CEECBLDY converts a string representing a date into a COBOL Integer format, which is the number of days since 1 January 1601. This service is similar to CEEDAYS, except that it provides a string in COBOL Integer format, which is compatible with ANSI intrinsic functions. Use CEECBLDY to access the century window of Language Environment and to perform date calculations with COBOL intrinsic functions for programs compiled with the INTDATE(ANSI) compiler option.

Call CEECBLDY only from COBOL programs that use the returned value as input for COBOL intrinsic functions. You should not use the returned value with other Language Environment callable services, nor should you call CEECBLDY from any non-COBOL programs. Unlike CEEDAYS, there is no inverse function for CEECBLDY, because it is only for COBOL users who want to use the Language Environment century window service together with COBOL intrinsic functions for date calculations. The inverse function for CEECBLDY is provided by the DATE-OF-INTEGER and DAY-OF-INTEGER intrinsic functions.

To handle dates earlier than 1601, add 4000 to each year, convert to Integer, calculate, subtract 4000 from the result, and then convert back to character format. By default, 2-digit years lie within the 100-year range starting 80 years prior to the system date. Thus, in 1995, all 2-digit years represent dates between 1915 and 2014, inclusive. You can change this default range by using the CEESCEN callable service.

Syntax

```
►►──CEECBLDY──(──input_char_date──,──picture_string──,──output_ANSI_date──,────►

►──)──────────────────────────────────────────────────────────────────────────►◄
```

input_char_date **(input)**
: A halfword length-prefixed character string (VSTRING) representing a date or timestamp, in a format conforming to that specified by *picture_string*. The character string must contain between 5 and 255 characters, inclusive. *input_char_date* can contain leading or trailing blanks. Parsing for a date begins with the first nonblank character (unless the picture string itself contains leading blanks, in which case CEECBLDY skips exactly that many positions before parsing begins). After parsing a valid date, as determined by the format of the date specified in *picture_string*, CEECBLDY ignores all remaining

characters. Valid dates range between and include 01 January 1601 to 31 December 9999. See Table 33 on page 487 for a list of valid picture character terms that can be specified in *input_char_date*.

picture_string **(input)**
A halfword length-prefixed character string (VSTRING), indicating the format of the date specified in *input_char_date*. Each character in the *picture_string* corresponds to a character in *input_char_date*. For example, if you specify MMDDYY as the *picture_string*, CEECBLDY reads an *input_char_date* of 060288 as 02 June 1988. If delimiters such as the slash (/) appear in the picture string, leading zeros can be omitted. For example, the following calls to CEECBLDY would each assign the same value, 148155 (02 June 1988), to COBINTDATE:

```
MOVE '6/2/88' TO DATEVAL.
MOVE 'MM/DD/YY' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);

MOVE '06/02/88' TO DATEVAL.
MOVE 'MM/DD/YY' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);

MOVE '060288' TO DATEVAL.
MOVE 'MMDDYY' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);

MOVE '88154' TO DATEVAL.
MOVE 'YYDDD' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);
```

Whenever characters such as colons or slashes are included in the *picture_string* (such as `HH:MI:SS YY/MM/DD`), they count as placeholders but are otherwise ignored. See Table 33 on page 487 for a list of valid picture character terms and Table 34 on page 488 for examples of valid picture strings.

If *picture_string* includes a Japanese Era symbol `<JJJJ>`, the `YY` position in *input_char_date* is replaced by the year number within the Japanese Era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 34 on page 488 for an additional example. See also Table 35 on page 488 for a list of Japanese Eras supported by CEEDATE.

If *picture_string* includes era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *input_char_date* is replaced by the year number within the era. See Table 34 on page 488 for an additional example.

output_Integer_date **(output)**
A 32-bit binary integer representing the COBOL Integer date, the number of days since 31 December 1600. For example, 16 May 1988 is day number 141485. If *input_char_date* does not contain a valid date, *output_Integer_date* is set to 0 and CEECBLDY terminates with a non-CEE000 symbolic feedback code. Date calculations are performed easily on the *output_Integer_date*, because *output_Integer_date* is an integer. Leap year and end-of-year anomalies do not affect the calculations.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can arise from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EB	3	2507	Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.
CEE2EC	3	2508	The date value passed to CEEDAYS or CEESECS was invalid.
CEE2ED	3	2509	The era passed to CEEDAYS or CEESECS was not recognized.
CEE2EH	3	2513	The input date passed in a CEEISEC, CEEDAYS, or CEESECS call was not within the supported range.
CEE2EL	3	2517	The month value in a CEEISEC call was not recognized.
CEE2EM	3	2518	An invalid picture string was specified in a call to a date/time service.
CEE2EO	3	2520	CEEDAYS detected non-numeric data in a numeric field, or the date string did not match the picture string.
CEE2EP	3	2521	The (<JJJJ>) or (<CCCC>) year-within-era value passed to CEEDAYS or CEESECS was zero.

Usage notes

- The probable cause for receiving message number 2518 is a picture string that contains an invalid DBCS string. You should verify that the data in the picture string is correct.
- z/OS UNIX consideration—In multithread applications, CEECBLDY affects only the calling thread.

For more information

- See the INTDATE COBOL compiler installation option in 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) for information about how to get ANSI integer values from COBOL Intrinsic Functions that are compatible with the Language Environment callable services CEEDAYS and CEEDATE.
- See "CEESCEN—Set the century window" on page 394 for more information about the CEESCEN callable service.
- See Table 33 on page 487 for a list of valid picture character terms that can be specified in *input_char_date*.

Examples

1. Following is an example of CEECBLDY called by COBOL.

```
 CBL LIB,QUOTE
*Module/File Name: IGZTCBLD
*************************************************
**                                             **
** Function: Invoke CEECBLDY callable service  **
** to convert date to COBOL Integer format.    **
** This service is used when using the         **
** Lang. Environ. Century Window                **
** mixed with COBOL Intrinsic Functions.       **
**                                             **
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLDY.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  CHRDATE.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                         of CHRDATE.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                         of  PICSTR.
 01  INTEGER                 PIC S9(9) BINARY.
 01  NEWDATE                 PIC 9(8).
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                 REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION.

 PARA-CBLDAYS.
*************************************************
** Specify input date and length               **
*************************************************
     MOVE 25 TO Vstring-length of CHRDATE.
     MOVE "1 January 00"
         to Vstring-text of CHRDATE.
*************************************************
** Specify a picture string that describes     **
** input date, and set the string's length.    **
*************************************************
     MOVE 23 TO Vstring-length of PICSTR.
     MOVE "ZD Mmmmmmmmmmmmmmz YY"
               TO Vstring-text of PICSTR.

*************************************************
** Call CEECBLDY to convert input date to a    **
** COBOL Integer date                          **
*************************************************
     CALL "CEECBLDY" USING CHRDATE, PICSTR,
                           INTEGER, FC.

*************************************************
** If CEECBLDY runs successfully, then compute **
**     the date of the 90th day after the      **
**     input date using Intrinsic Functions    **
*************************************************
     IF CEE000 of FC  THEN
         COMPUTE INTEGER = INTEGER + 90
         COMPUTE NEWDATE = FUNCTION
             DATE-OF-INTEGER (INTEGER)
         DISPLAY NEWDATE
             " is ANSI day: " INTEGER
     ELSE
```

```
              DISPLAY "CEECBLDY failed with msg "
                  Msg-No of FC UPON CONSOLE
              STOP RUN
          END-IF.

          GOBACK.
```

CEECMI—Store and load message insert data

CEECMI copies message insert data and loads the address of that data into the Instance Specific Information (ISI) associated with the condition being processed. CEECMI also allocates storage for the ISI, if necessary. The number of ISIs per thread is determined by the MSGQ runtime option. ISIs are released when the value specified in the MSGQ runtime option is exceeded. The least recently used ISI is overwritten.

If you plan on using a routine that signals a new condition with a call to the CEESGL callable service, you should first call CEECMI to copy any insert information into the ISI associated with the condition.

Syntax

```
►►──CEECMI──(──cond_rep──,── insert_seq_num──,──insert_data──,──fc──)──────────►◄
```

cond_rep **(input/output)**
A condition token that defines the condition for which the q_data_token is retrieved.

insert_seq_num **(input)**
A 4-byte integer that contains the insert sequence number (such as insert 1 insert 2). It corresponds to an insert number specified with an `ins` tag in the message source file created by the CEEBLDTX EXEC.

insert_data **(input)**
A halfword-prefixed length string that represents the insert data. The entire length described in the halfword prefix is used without truncation. DBCS strings must be enclosed within shift-out (X'0E') and shift-in (X'0F') characters. The maximum size for an individual insert data item is 254 bytes.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can arise from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0EB	3	0459	Not enough storage was available to create a new instance specific information block.
CEE0EC	1	0460	Multiple instances of the condition token with message number *message-number* and facility ID *facility-id* were detected.

Code	Severity	Message number	Message text
CEE0ED	3	0461	The maximum number of unique message insert blocks was reached. This condition token had its I_S_info field set to 1.
CEE0EE	3	0462	Instance specific information for the condition token with message number *message-number* and facility ID *facility-id* could not be found.
CEE0EF	3	0463	The maximum size for an insert data item was exceeded.
CEE0H9	3	0553	An internal error was detected in creating the inserts for a condition.

Usage notes

- z/OS UNIX consideration—In multithread applications, CEECMI applies to message insert data for only the calling thread.

For more information

- See "MSGQ" on page 56 for more information about the MSGQ runtime option.
- For more information about CEEBLDTX, see *z/OS Language Environment Programming Guide*.

Examples

1. Following is an example of CEECMI called by C/C++.

```
/*Module/File Name: EDCCMI    */
   /************************************************
    **                                              *
    ** FUNCTION: CEENCOD - set up a condition token  *
    **         : CEECMI  - store and load message    *
    **                     insert data               *
    **         : CEEMSG  - retrieve, format, and     *
    **                     dispatch a message to     *
    **                     message file              *
    **                                               *
    **   This example illustrates the invocation of  *
    **   the Lang. Environ. message services to      *
    **    store and load message insert data.        *
    **   The resulting message and insert is written *
    **   to the Lang. Environ. MSGFILE ddname.       *
    **                                               *
    ************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

void main ()
{
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;
  _VSTRING insert;
  _FEEDBACK ctok;
  _FEEDBACK fbcode;
  _INT4 MSGFILE;
  _INT4 insert_no ;
  /* Condition Token Declarations */
  /**************************************************
   * EXMPLMSG is a token that represents message     *
  * number 10 in a user message file constructed     *
  * using the CEEBLDTXT facility.                     *
  * Message 10 is designed to allow one insert.       *
  ****************************************************/
 insert.length = 18;
 memcpy(insert.string ,"<CEPGCMI's insert>",
        insert.length);
 /*give ctok value of hex 0000000A40E7D4D700000000   */
 /*sev = 0  msgno =  10 facid = XMP                  */
  c_1 = 0;
  c_2 = 10;
  cond_case = 1;
  sev = 0;
  control = 0;
  memcpy(facid,"XMP",3);
  isi = 0;

 /*********************************************/
 /* Call CEENCOD to set-up a condition token */
 /*********************************************/
  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&ctok,&fbcode);
  if ( _FBCHECK ( fbcode , CEE000 ) != 0 )
   printf("CEENCOD failed with message number %d\n",
          fbcode.tok_msgno);
/*******************************************/
 /* Call CEECMI to create a message insert */
 /*******************************************/
  CEECMI(&ctok, &insert_no, &insert, &fbcode);
  if ( _FBCHECK ( fbcode , CEE000 ) != 0 )
   printf("CEECMI failed with message number %d\n",
          fbcode.tok_msgno);

 /*******************************************/
 /* Call CEEMSG to issue the message       */
 /*******************************************/
  CEEMSG(&ctok, &MSGFILE , &fbcode);
  if ( _FBCHECK ( fbcode , CEE000 ) != 0 )
   printf("CEEMSG failed with message number %d\n",
          fbcode.tok_msgno);
}
```

2. Following is an example of CEECMI called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTCMI
      ***********************************************
      **                                            *
      ** Function: CEECMI  - Store and load message *
      **                     insert data            *
      **         : CEENCOD - Construct a condition  *
      **                     token                  *
      **         : CEEMSG  - Dispatch a Message.    *
      **                                            *
      **   This example illustrates the invocation  *
      **   of the Lang. Environ. message services to*
      **    store and load message insert data. *
      **   CEENCOD is called to construct a token   *
      **   for a user defined message (message 10)  *
      **   in a user message file.                  *
      **   CEECMI is called to insert text into     *
      **   message 10. The resulting message and    *
      **   insert is written to the MSGFILE.        *
      **                                            *
      ***********************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLCMI.
       DATA DIVISION.
```

```
 WORKING-STORAGE SECTION.
 01  INSERTNO               PIC S9(9) BINARY.
 01  CTOK                   PIC X(12).
 01  FBCODE.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity       PIC S9(4) BINARY.
             04  Msg-No         PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code     PIC S9(4) BINARY.
             04  Cause-Code     PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info           PIC S9(9) BINARY.
 01  MSGDEST                PIC S9(9) BINARY.
 01  SEV                    PIC S9(4) BINARY.
 01  MSGNO                  PIC S9(4) BINARY.
 01  CASE                   PIC S9(4) BINARY.
 01  SEV2                   PIC S9(4) BINARY.
 01  CNTRL                  PIC S9(4) BINARY.
 01  FACID                  PIC X(3).
 01  ISINFO                 PIC S9(9) BINARY.
 01  VSTRING.
     05  INSERT-TXTL        PIC S9(4) BINARY.
     05  INSERT-TXT         PIC X(80).
 PROCEDURE DIVISION.
 PARA-CEPGCMI.
************************************************
*   Set up token fields for creation of a      *
*   condition token for the user defined       *
*   message file and message number.           *
************************************************
     MOVE   0 TO SEV.
     MOVE  10 TO MSGNO.
     MOVE   1 TO CASE.
     MOVE   0 TO SEV2.
     MOVE   0 TO CNTRL.
     MOVE "XMP" TO FACID.
     MOVE   0 TO ISINFO.
************************************************
*  Call CEENCOD to construct a condition token *
************************************************
     CALL "CEENCOD" USING SEV, MSGNO, CASE,
                          SEV2, CNTRL, FACID,
                          ISINFO, CTOK, FBCODE.
     IF NOT CEE000 of FBCODE  THEN
         DISPLAY "CEENCOD failed with msg"
             Msg-No of FBCODE UPON CONSOLE
         STOP RUN
     END-IF.
***********************************************
*   Call CEECMI to store and load message     *
*   insert 1.                                 *
***********************************************
     MOVE "<CEPGCMI""s insert>" TO INSERT-TXT.
     MOVE 19 TO INSERT-TXTL.
     MOVE 1 TO INSERTNO.
     CALL "CEECMI" USING CTOK, INSERTNO, VSTRING.
***********************************************
*   Call CEEMSG to write message to MSGFILE   *
***********************************************
     MOVE 2 TO MSGDEST.
     CALL "CEEMSG" USING CTOK, MSGDEST, FBCODE.
     IF  NOT CEE000 of FBCODE  THEN
         DISPLAY "CEEMSG failed with msg "
```

```
                 Msg-No of FBCODE UPON CONSOLE
             STOP RUN
         END-IF.

         GOBACK.
```

3. Following is an example of CEECMI called by PL/I.

```
*PROCESS MACRO;
 IBMCMI: Proc Options(Main);

   /*Module/File Name: IBMCMI                        */
   /*************************************************
    **                                               *
    ** FUNCTION  : CEECMI  - store and load message  *
    **                       insert data             *
    **           : CEEMSG  - retrieve, format, and   *
    **                       dispatch a message to   *
    **                       message file            *
    **                                               *
    **   This example illustrates the invocation of  *
    **   LE/370 message services to store and load   *
    **   message insert data. The resulting message  *
    **   and insert are written to the MSGFILE.      *
    **                                               *
    *************************************************/
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
    DECLARE INSERT     CHAR(255) VARYING;
    DCL 01 CTOK,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);
    DCL 01 FBCODE,                 /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);
    DECLARE MSGFILE    REAL FIXED BINARY(31,0);
    DECLARE INSERT_NO  REAL FIXED BINARY(31,0);

    /************************************************/
    /* Ctok is initialized in the DECLARE statement  */
    /* to message 10 in a user message file          */
    /* constructed using the CEEBLDTX tool.          */
    /* Message 10 is designed to allow one insert.   */
    /* The message facility ID is XMP.               */
    /************************************************/
    insert = '<CEPGCMI's insert>';
    insert_no = 1;

    /*****************************************/
    /* Call CEECMI to create a message insert */
    /*****************************************/
    Call CEECMI(ctok, insert_no, insert, *);

    /*****************************************/
```

```
      /* Call CEEMSG to issue the message       */
      /*****************************************/
      MSGFILE = 2;
      Call CEEMSG(ctok, MSGFILE, *);

   End IBMCMI;
```

CEECRHP—Create new additional heap

CEECRHP lets you define additional heaps. It returns a unique *heap_id*. The heaps defined by CEECRHP can be used just like the initial heap (*heap_id=0*), below heap, and anywhere heap. Unlike the heaps created by these heap services, all heap elements within an additional heap can be quickly freed by a single call to CEEDSHP (discard heap). The number of heaps supported by Language Environment is limited only by the amount of virtual storage available.

Syntax

```
►►—CEECRHP—(—heap_id—,—initial_size—,—increment—,—options—,—fc—————►

►—)————————————————————————————————————————————————————————————————►◄
```

heap_id **(output)**
A fullword binary signed integer. *heap_id* is the heap identifier of the created heap. If a new heap cannot be created, the value of *heap_id* remains undefined. Storage obtained from *heap_id*s 79 and 80 is set to binary 0 independent of any initialization value specified by the STORAGE option.

initial_size **(input)**
A fullword binary signed integer. *initial_size* is the initial amount of storage, in bytes, allocated for the new heap. *initial_size* is rounded up to the nearest increment of 4096 bytes. If *initial_size* is specified as 0, then the *init_size* specified in the HEAP runtime option is used. If no HEAP runtime option was provided and *initial_size* is specified as 0, CEECRHP uses the system-level or region-level default.

increment **(input)**
A fullword binary signed integer. When it is necessary to enlarge the heap to satisfy an allocation request, *increment* represents the number of bytes by which the heap is extended. *increment* is rounded up to the nearest 4096 bytes. If *increment* is specified as 0, then the *incr_size* specified in the HEAP run time option is used. If no HEAP runtime option was provided and *increment* equals 0, CEECRHP uses the installation default.

options **(input)**
A fullword binary signed integer. *options* are specified with the decimal codes, as shown in Table 25.

Table 25. HEAP attributes based on the setting of the options parameter

Option setting	HEAP attributes
00	Use same attributes as the initial heap (copy them from the HEAP runtime option)
01	HEAP(,,,FREE) (location inherited from HEAP runtime option)

Table 25. HEAP attributes based on the setting of the options parameter (continued)

Option setting	HEAP attributes
70	HEAP(,,,KEEP) (location inherited from HEAP runtime option)
71	HEAP(,,ANYWHERE,KEEP)
72	HEAP(,,ANYWHERE,FREE)
73	HEAP(,,BELOW,KEEP)
74	HEAP(,,BELOW,FREE)
75	HEAP(,,ANYWHERE,) (disposition inherited from the HEAP runtime option)
76	HEAP(,,BELOW,) (disposition inherited from the HEAP runtime option)
77	HEAP(,,ANYWHERE,KEEP) (all heap storage obtained using this heap_id is allocated on a 4K boundary)
78	HEAP(,,ANYWHERE,FREE) (all heap storage obtained using this heap_id is allocated on a 4K boundary)
79	HEAP(,,ANYWHERE,KEEP) (all heap storage obtained using this heap_id is set to binary 0 when allocated using CEEGTST)
80	HEAP(,,ANYWHERE,FREE) (all heap storage obtained using this heap_id is set to binary 0 when allocated using CEEGTST)

`fc (output)`

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can arise from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P4	3	0804	The initial size value supplied in a create heap request was unsupported.
CEE0P5	3	0805	The increment size value supplied in a create heap request was unsupported.
CEE0P6	3	0806	The options value supplied in a create heap request was unrecognized.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage request.

Usage notes

- z/OS UNIX consideration—CEECRHP applies to the enclave.
- The heapid can only be used by the TCB on which the CEECRHP request was issued. Using the heapid on other TCBs is not supported and will generate unpredictable results.

For more information

- See “CEEDSHP—Discard heap” on page 256 for more information about the CEEDSHP callable service.
- See “HEAP” on page 36 for more information about the HEAP runtime option and IBM-supplied defaults.

Examples

1. Following is an example of CEECRHP called by C/C++.

```
/*Module/File Name: EDCCRHP   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _INT4 heapid, size, increment, options;
   _FEEDBACK fc;
 /*  .
     .
     .  */
   heapid = 0;        /* heap identifier is set */
                      /* by CEECRHP */
   size = 4096;       /* initial size of heap  (in */
                      /* bytes) */
   increment = 4096;  /* increment to extend heap by */
   options = 72;      /* set up heap as */
                      /* (,,ANYWHERE,FREE) */

   /* create heap using CEECRHP */
   CEECRHP(&heapid,&size,&increment,&options,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful)  */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEECRHP failed with message number %d\n",
             fc.tok_msgno);
      exit(99);
   }
 /*  .
     .
     .  */
   /* discard the heap that was previously created  */
   /* using CEECRHP */
   CEEDSHP(&heapid,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000) != 0 ) {
      printf("CEEDSHP failed with message number %d\n",
             fc.tok_msgno);
      exit(99);
   }
 /*  .
     .
     .  */
}
```

2. Following is an example of CEECRHP called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTCRHP
      ***********************************************
      **                                           **
      ** Function: CEECRHP - create new additional **
      **                     heap                  **
      **                                           **
      ***********************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLCRHP.
       DATA DIVISION.
```

```
 WORKING-STORAGE SECTION.
 01  HEAPID                 PIC S9(9) BINARY.
 01  HPSIZE                 PIC S9(9) BINARY.
 01  INCR                   PIC S9(9) BINARY.
 01  OPTS                   PIC S9(9) BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION.
*************************************************
** Specify 0 for HEAPID, and heap id will be   **
**     set by CEECRHP.                         **
** Heap size and increment will each be        **
**     4096 bytes.                             **
** Specify 00 for OPTS, and HEAP attributes    **
**     will be inherited from the initial heap **
**     (copied from the HEAP runtime option). **
*************************************************
     MOVE 0 TO HEAPID.
     MOVE 4096 TO HPSIZE.
     MOVE 4096 TO INCR.
     MOVE 00 TO OPTS.

     CALL "CEECRHP" USING HEAPID, HPSIZE,
                          INCR, OPTS, FC.
     IF CEE000 of FC  THEN
         DISPLAY "Created heap number " HEAPID
              " which is " HPSIZE " bytes long"
     ELSE
         DISPLAY "CEECRHP failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEECRHP called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMCRHP                      */

 /*************************************************/
 /**                                              */
 /** Function: CEECRHP - create new additional    */
 /**                     heap                     */
 /**                                              */
 /** In this example, CEECRHP is called to set up */
 /** a new additional heap of 4096 bytes. Each    */
 /** time the heap needs to be extended, an       */
 /** increment of 4096 bytes will be added.       */
 /**                                              */
 /*************************************************/
 PLICRHP: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
    DCL HEAPID  REAL FIXED BINARY(31,0) ;
    DCL HPSIZE  REAL FIXED BINARY(31,0) ;
```

```
      DCL INCR    REAL FIXED BINARY(31,0) ;
      DCL OPTS    REAL FIXED BINARY(31,0) ;
      DCL 01 FC,                    /* Feedback token */
             03 MsgSev    REAL FIXED BINARY(15,0),
             03 MsgNo     REAL FIXED BINARY(15,0),
             03 Flags,
                05 Case      BIT(2),
                05 Severity  BIT(3),
                05 Control   BIT(3),
             03 FacID     CHAR(3),    /* Facility ID */
             03 ISI   /* Instance-Specific Information */
                          REAL FIXED BINARY(31,0);

      HEAPID = 0;    /* HEAPID will be set and     */
                     /* returned by CEECRHP        */
      HPSIZE = 4096; /* Initial size of heap,      */
                     /* in bytes                   */
      INCR = 4096;   /* Number of bytes to extend  */
                     /* heap by                    */
      OPTS = 00;     /* Set up heap with the same  */
                     /* attributes as the          */
                     /* initial heap (HEAPID = 0)  */

      /* Call CEECRHP to set up new heap  */
      CALL CEECRHP ( HEAPID, HPSIZE, INCR, OPTS, FC );
      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST( 'Created heap number ' || HEAPID
            || ' consisting of ' || HPSIZE || ' bytes' );
         END;
      ELSE  DO;
         DISPLAY( 'CEECRHP failed with msg '
            || FC.MsgNo );
         STOP;
         END;

   END PLICRHP;
```

CEECZST—Reallocate (change size of) storage

CEECZST changes the size of a previously allocated heap element. The *address* parameter points to the beginning of the heap element. The *new_size* parameter gives the new size of the heap element, in bytes. The contents of the heap element are unchanged up to the shorter of the new and old sizes.

The CEECZST service returns a pointer to the reallocated heap element. It can move the storage location of the heap element. As a result, the *address* parameter passed to CEECZST is not necessarily the same as the value returned.

Because the new storage might be allocated at a different location from the existing allocation, any pointers (specifically any addresses) that referred to the old storage become invalid. Continued use of such dangling pointers gives unpredictable, and almost certainly incorrect, results.

The heap identifier is inferred from the address. The new storage block is allocated from the same heap that contained the old block.

The contents of the old storage are preserved in the following manner:

- If *new_size* is greater than the old size, the entire contents of the old storage block are copied to the new block. The remaining bytes in the new element are left uninitialized unless an initialization suboption value was specified for the heap in the STORAGE option.

- If *new_size* is less than the old size, the contents of the old block are truncated to the size of the new block.
- If *new_size* is equal to the old size, no operations are performed; a successful feedback code is returned.

Syntax

```
►►──CEECZST──(──address──,──new_size──,──fc──)──────────────────────────►◄
```

address **(input/output)**
A fullword address pointer. On input, this parameter contains an address returned by a previous CEEGTST call. On output, the address of the first byte of the newly allocated storage is returned in this parameter.

new_size **(input)**
A fullword binary signed integer. *new_size* is the number of bytes of storage to be allocated for the new heap element.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P8	3	0808	Storage size in a get storage request or a reallocate request was not a positive number.
CEE0PA	3	0810	The storage address in a free storage request was not recognized, or heap storage control information was damaged.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage request.

Usage notes

- Storage that is reallocated maintains the same mark/release status as the old storage block. If the old storage block was marked, the new storage block carries the same mark and is released by a release operation that specifies that mark.
- z/OS UNIX consideration—CEECZST applies to the enclave.
- The _CEE_REALLOC_CONTROL environment variable provides additional levels of storage control, which can aid CEECZST to more efficiently use storage. For more information about _CEE_REALLOC_CONTROL, see*z/OS XL C/C++ Programming Guide*.

For more information

- See “STORAGE” on page 79 for more information about the STORAGE runtime option.

- For information about CEEGTST, see “CEEGTST—Get heap storage” on page 313.

Examples

1. Following is an example of CEECZST called by C/C++.

```
/*Module/File Name: EDCCZST   */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>
int main(void) {
   _INT4 heapid, size;
   _POINTER address;
   _FEEDBACK fc;
  /* .
     .
     .  */
   heapid = 0;    /* get storage from initial heap */
   size = 4000;   /* number of bytes of heap storage */

   /* obtain the storage using CEEGTST */
   CEEGTST(&heapid,&size,&address,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEGTST failed with message number %d\n",
             fc.tok_msgno);
      exit(99);
   }
  /* .
     .
     .  */
   size = 2000;  /* new size of storage element */

   /* change the size of the storage element */
   CEECZST(&address,&size,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEECZST failed with message number %d\n",
              fc.tok_msgno);
      exit(99);
   }
  /* .
     .
     .  */
   /* free the storage that was previously obtained */
   /* using CEEGTST */
   CEEFRST(&address,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEFRST failed with message number %d\n",
              fc.tok_msgno);
      exit(99);
   }
  /* .
     .
     .  */
}
```

2. Following is an example of CEECZST called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTCZST
      ************************************************
      **                                            **
      ** Function: CEECZST - reallocate storage     **
      **                                            **
      ** In this example, CEEGTST is called to      **
      ** request storage from HEAPID = 0, and       **
      ** CEECZST is called to change the size of    **
      ** that storage request.                      **
      **                                            **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLCZST.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01   HEAPID                 PIC S9(9) BINARY.
       01   HPSIZE                 PIC S9(9) BINARY.
       01   ADDRSS                 POINTER.
       01   NEWSIZE                PIC S9(9) BINARY.
       01   FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLGTST.
      *************************************************
      ** Specify 0 to get storage from the initial   **
      **     heap. Specify 4000 to get 4000 bytes of **
      **     storage.                                **
      *************************************************
           MOVE 0 TO HEAPID.
           MOVE 4000 TO HPSIZE.

      *************************************************
      ** Call CEEGTST to obtain storage.             **
      *************************************************
           CALL "CEEGTST" USING HEAPID, HPSIZE,
                                ADDRSS, FC.

      *************************************************
      ** If CEEGTST runs successfully, display result**
      *************************************************
           IF CEE000 OF FC THEN
               DISPLAY " " HPSIZE
                   " bytes have been allocated."
           ELSE
               DISPLAY "CEEGTST failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
      *************************************************
      ** Specify a new size of 2000 bytes.           **
      *************************************************
           MOVE 2000 TO NEWSIZE.

      *************************************************
```

```
     ** Call CEECZST to change the size of the      **
     ** storage allocated in the call to CEEGTST.   **
     *************************************************
          CALL "CEECZST" USING ADDRSS, NEWSIZE, FC.

     *************************************************
     ** If CEECZST runs successfully, display result**
     *************************************************
          IF CEE000 OF FC THEN
              DISPLAY
                  "The storage element now contains "
                  NEWSIZE " bytes."
          ELSE
              DISPLAY "CEEGTST failed with msg "
                  Msg-No of FC UPON CONSOLE
              STOP RUN
          END-IF.

          GOBACK.
```

3. Following is an example of CEECZST called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMCZST                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEECZST - reallocate storage         **/
 /**                                                **/
 /** In this example, CEEGTST is called to request **/
 /** storage from HEAPID = 0, and CEECZST is called**/
 /** to change the size of that storage request.   **/
 /**                                                **/
 /**                                                **/
 /****************************************************/
 PLICZST: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL HEAPID   REAL FIXED BINARY(31,0) ;
    DCL STGSIZE  REAL FIXED BINARY(31,0) ;
    DCL ADDRSS1  POINTER;
    DCL 01 FC1,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    DCL ADDRSS2  POINTER;
    DCL NEWSIZE  REAL FIXED BINARY(31,0) ;
    DCL 01 FC2,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    HEAPID = 0;    /* get storage from initial heap */
    STGSIZE = 4000; /* get 4000 bytes of storage    */
```

```
        /* Call CEEGTST to obtain the storage          */
        CALL CEEGTST ( HEAPID, STGSIZE, ADDRSS1, FC1 );
        IF  FBCHECK( FC1, CEE000)  THEN  DO;
           PUT SKIP LIST( 'Obtained ' || STGSIZE
              || ' bytes of storage at location '
              || DECIMAL( UNSPEC( ADDRSS1 ) )
              || ' from heap ' || HEAPID );
           END;
        ELSE  DO;
           DISPLAY( 'CEEGTST failed with msg '
              || FC1.MsgNo );
           STOP;
           END;

        NEWSIZE = 2000;
               /* change size of HEAPID 0 to 2000 bytes */

        /* Call CEECZST to change the size of storage   */
        ADDRSS2 = ADDRSS1;
        CALL CEECZST ( ADDRSS2, NEWSIZE , FC2 );
        IF  FBCHECK( FC2, CEE000)  THEN  DO;
           PUT SKIP LIST( 'Obtained ' || NEWSIZE
              || ' bytes of storage at location '
              || DECIMAL( UNSPEC( ADDRSS1 ) ) );
           PUT SKIP LIST( 'Original ' || STGSIZE
              || ' bytes of storage at location '
              || DECIMAL( UNSPEC( ADDRSS1 ) )
              || ' no longer valid' );
           END;
        ELSE  DO;
           DISPLAY( 'CEECZST failed with msg '
              || FC2.MsgNo );
           STOP;
           END;

     END PLICZST;
```

CEEDATE—Convert Lilian date to character format

CEEDATE converts a number representing a Lilian date to a date written in character format. The output is a character string, such as 1993/09/09.

Do not use CEEDATE in combination with COBOL intrinsic functions.

The inverse of CEEDATE is CEEDAYS, which converts character dates to the Lilian format.

CEEDATE is affected only by the country code setting of the COUNTRY runtime option or CEE3CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax

```
►►──CEEDATE──(──input_Lilian_date──,──picture_string──,──output_char_date──────►

►──,──fc──)──────────────────────────────────────────────────────────────────►◄
```

`input_Lilian_date` **(input)**
A 32-bit integer representing the Lilian date. The Lilian date is the number of days since 14 October 1582. For example, 16 May 1988 is Lilian day number 148138. The valid range of Lilian dates is 1 to 3,074,324 (15 October 1582 to 31 December 9999).

`picture_string` **(input)**
A halfword length-prefixed character string (VSTRING), representing the desired format of *output_char_date*, for example `MM/DD/YY`. Each character in *picture_string* represents a character in *output_char_date*. If delimiters such as the slash (/) appear in the picture string, they are copied to *output_char_date*.

See Table 33 on page 487 for a list of valid picture characters, and Table 34 on page 488 for examples of valid picture strings.

If *picture_string* is null or blank, CEEDATE gets *picture_string* based on the current value of the COUNTRY runtime option. For example, if the current value of the COUNTRY runtime option is US (United States), the date format would be `MM/DD/YY`. If the current COUNTRY value is FR (France), the date format would be `MM/DD/YY  HH:MM:SS AM (or PM)`, for example: `09/09/93 4:56:29 PM`. This default mechanism makes it easy for translation centers to specify the preferred date, and for applications and library routines to use this format automatically.

If *picture_string* includes a Japanese Era symbol `<JJJJ>`, the `YY` position in *output_char_date* is replaced by the year number within the Japanese Era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 34 on page 488 for an additional example. Also see Table 35 on page 488 for a list of Japanese Eras supported by CEEDATE.

If *picture_string* includes a era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *output_char_date* is replaced by the year number within the era. See Table 34 on page 488 for an example.

`output_char_date` **(output)**
A fixed-length 80-character string (VSTRING), is the result of converting *input_Lilian_date* to the format specified by *picture_string*. See Table 26 on page 235 for sample output dates. If *input_Lilian_date* is not valid, *output_char_date* is set to all blanks. CEEDATE terminates with a non-CEE000 symbolic feedback code.

`fc` **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EG	3	2512	The Lilian date value was not within the supported range.
CEE2EM	3	2518	An incorrect picture string was specified.
CEE2EQ	3	2522	<JJJJ>, <CCCC> or <CCCCCCCC> was used in a picture string passed to CEEDATE, but the Lilian date value was not within the supported range. The Era could not be determined.
CEE2EU	2	2526	The date string returned by CEEDATE was truncated.

Usage notes

- The probable cause for receiving message number 2518 is a picture string that contains a DBCS string that is not valid. You should verify that the data in the picture string is correct.
- To create a null VSTRING, set the length to zero; the content of the text portion does not matter. To create a blank VSTRING, any length greater than zero can be used; the content of the text portion must be spaces or blanks.
- z/OS UNIX consideration—In multithread applications, CEEDATE applies to the enclave.

For more information

- See the INTDATE COBOL compiler installation option in 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) for information about how to get Lilian integer values from COBOL intrinsic functions that are compatible with the Language Environment callable services CEEDAYS and CEEDATE.
- See "CEEDAYS—Convert date to Lilian format" on page 241 for more information about the CEEDAYS callable service.
- See "COUNTRY" on page 22 for more information about the COUNTRY runtime option.
- See "CEEFMDA—Get default date format" on page 268 for information about how to get the default format for a given country code.

Examples

1. Following is an example of CEEDATE called by C/C++.

```
/*Module/File Name: EDCDATE   */

#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _INT4 lil_date = 139370;   /* May 14, 1964 */
   _VSTRING date_pic,date;
   _CHAR80 date_out;

   strcpy(date_pic.string,
     "The date is Wwwwwwwwwz, Mmmmmmmmmz ZD, YYYY");
   date_pic.length = strlen(date_pic.string);

   CEEDATE(&lil_date,&date_pic,date_out,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEDATE failed with message number %d\n",
              fc.tok_msgno);
      exit(2999);
   }
   printf("%.80s\n",date_out);
}
```

2. Following is an example of CEEDATE called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTDATE
      ************************************************
      **                                            **
** Function: CEEDATE - convert Lilian date to **
**                    character format        **
**                                            **
** In this example, a call is made to CEEDATE **
** to convert a Lilian date (the number of    **
** days since 14 October 1582) to a character **
** format (such as 6/22/88). The result is    **
** displayed.  The Lilian date is obtained    **
** via a call to CEEDAYS.                     **
**                                            **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLDATE.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  LILIAN                  PIC S9(9) BINARY.
 01  CHRDATE                 PIC X(80).
 01  IN-DATE.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                         of IN-DATE.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of PICSTR.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.
 PARA-CBLDAYS.
*************************************************
** Call CEEDAYS to convert date of 6/2/88 to   **
**     Lilian representation                   **
*************************************************
     MOVE 6 TO Vstring-length of IN-DATE.
     MOVE "6/2/88" TO Vstring-text of IN-DATE(1:6).
     MOVE 8 TO Vstring-length of PICSTR.
     MOVE "MM/DD/YY" TO Vstring-text of PICSTR(1:8).
     CALL "CEEDAYS" USING IN-DATE, PICSTR,
                          LILIAN, FC.

*************************************************
** If CEEDAYS runs successfully, display result**
*************************************************
     IF  CEE000 of FC  THEN
         DISPLAY Vstring-text of IN-DATE
             " is Lilian day: " LILIAN
     ELSE
         DISPLAY "CEEDAYS failed with msg "
             Msg-No of FC UPON CONSOLE
```

```
                STOP RUN
            END-IF.

       *************************************************
       ** Specify picture string that describes the   **
       **  desired format of the output from CEEDATE, **
       **  and the picture string's length.           **
       *************************************************
            MOVE 23 TO Vstring-length OF PICSTR.
            MOVE "ZD Mmmmmmmmmmmmmmz YYYY" TO
                         Vstring-text OF PICSTR(1:23).

       *************************************************
       ** Call CEEDATE to convert the Lilian date     **
       **     to  a picture string.                   **
       *************************************************
            CALL "CEEDATE" USING LILIAN, PICSTR,
                                 CHRDATE, FC.
       *************************************************
       ** If CEEDATE runs successfully, display result**
       *************************************************
            IF CEE000 of FC  THEN
                DISPLAY "Input Lilian date of " LILIAN
                    " corresponds to:  " CHRDATE
            ELSE
                DISPLAY "CEEDATE failed with msg "
                    Msg-No of FC UPON CONSOLE
                STOP RUN
            END-IF.

            GOBACK.
```

3. Following is an example of CEEDATE called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMDATE                   */
 /**********************************************/
 /**                                           */
 /** Function: CEEDATE - convert Lilian date to */
 /**                     character format      */
 /**                                           */
 /** In this example, a call is made to CEEDATE */
 /** to convert a date in the Lilian format     */
 /** (the number of days since 14 October 1582) */
 /** to a date in character format. This date   */
 /** is then printed out.                      */
 /**                                           */
 /******************************************** */
 PLIDATE: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL LILIAN  REAL FIXED BINARY(31,0) ;
    DCL PICSTR  CHAR(255) VARYING;
    DCL CHRDATE CHAR(80) ;
    DCL 01 FC,                   /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI    * Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    LILIAN = 152385;  /* input date in Lilian format */
```

```
         /* picture string that describes how converted   */
         /* date is to be formatted                       */
         PICSTR = 'ZD Mmmmmmmmmmmmmmz YYYY';

         /* Call CEE3DATE to convert input Lilian date to */
         /* a date in the character format specified in   */
         /* PICSTR                                        */
         CALL CEEDATE ( LILIAN , PICSTR , CHRDATE , FC );

         /* Print results if call to CEEDATE succeeds     */
         IF  FBCHECK( FC, CEE000)  THEN  DO;
             PUT SKIP LIST( 'Lilian day ' || LILIAN
                || ' is equivalent to ' || CHRDATE );
            END;
         ELSE  DO;
            DISPLAY( 'CEEDATE failed with msg '
               || FC.MsgNo );
            STOP;
            END;

      END PLIDATE;
```

Table 26 shows the sample output from CEEDATE.

Table 26. Sample output of CEEDATE

input_Lilian_date	picture_string	output_char_date
148138	YY	88
	YYMM	8805
	YY-MM	88-05
	YYMMDD	880516
	YYYYMMDD	19880516
	YYYY-MM-DD	1988-05-16
	YYYY-ZM-ZD	1988-5-16
	JJJJ YY.MM.DD	Showa 63.05.16 (in a DBCS string)
	CCCC YY.MM.DD	Min Guo 77.05.16 (in a DBCS string)
148139	MM	05
	MMDD	0517
	MM/DD	05/17
	MMDDYY	051788
	MM/DD/YYYY	05/17/1988
	ZM/DD/YYYY	5/17/1988
148140	DD	18
	DDMM	1805
	DDMMYY	180588
	DD.MM.YY	18.05.88
	DD.MM.YYYY	18.05.1988
	DD Mmm YYYY	18 May 1988
148141	DDD	140
	YYDD	88148
	YY.DDD	88.140
	YYYY.DDD	1988.140
148142	YY/MM/DD HH:MI:SS.99	88/05/20 00:00:00.00
	YYYY/ZM/ZD ZH:MI AP	1988/5/20 0:00 AM

Table 26. Sample output of CEEDATE (continued)

input_Lilian_date	picture_string	output_char_date
148143	`WWW., MMM DD, YYYY`	`SAT., MAY 21, 1988`
	`Www.,  Mmm DD, YYYY`	`Sat., May 21, 1988`
	`Wwwwwwwwww  Mmmmmmmmmm DD, YYYY`	`Saturdayƀƀ, Mayƀƀƀƀƀƀ 21, 1988`
	`Wwwwwwwwwz, Mmmmmmmmmz DD, YYYY`	`Saturday, May 21, 1988`

CEEDATM—Convert seconds to character timestamp

CEEDATM converts a number representing the number of seconds since 00:00:00 14 October 1582 to a character string format. The format of the output is a character string timestamp, for example: `1988/07/26 20:37:00`.

The inverse of CEEDATM is CEESECS, which converts a timestamp to number of seconds.

CEEDATM is affected only by the country code setting of the COUNTRY runtime option or CEE3CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax

```
►►──CEEDATM──(──input_seconds──,──picture_string──,──output_timestamp──,──────►

►──fc──)─────────────────────────────────────────────────────────────────────►◄
```

input_seconds **(input)**
: A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range of *input_seconds* is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999).

picture_string **(input)**
: A halfword length-prefixed character string (VSTRING), representing the desired format of *output_timestamp*, for example, `MM/DD/YY HH:MI AP`.

  Each character in the *picture_string* represents a character in *output_timestamp*. If delimiters such as a slash (/) appear in the picture string, they are copied as is to *output_timestamp*.

  See Table 33 on page 487 for a list of valid picture character terms and Table 34 on page 488 for examples of valid picture strings.

  If *picture_string* is null or blank, CEEDATM gets *picture_string* based on the current value of the COUNTRY runtime option. For example, if the current value of the COUNTRY runtime option is US (United States), the date-time format would be “`MM/DD/YY HH:MI:SS AP`”; if the current COUNTRY value is FR (France), however, the date-time format would be “`DD.MM.YYYY HH:MI:SS`”.

If *picture_string* includes the Japanese Era symbol `<JJJJ>`, the `YY` position in *output_timestamp* represents the year within Japanese Era. See Table 34 on page 488 for an example. See Table 35 on page 488 for a list of Japanese Eras supported by CEEDATM.

If *picture_string* includes era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *output_timestamp* represents the year within the era. See Table 34 on page 488 for an example.

output_timestamp **(output)**
A fixed-length 80-character string (VSTRING), that is the result of converting *input_seconds* to the format specified by *picture_string*. If necessary, the output is truncated to the length of *output_timestamp*. See Table 27 on page 241 for sample output.

If *input_seconds* is not valid, *output_timestamp* is set to all blanks and CEEDATM terminates with a non-CEE000 symbolic feedback code.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E9	3	2505	The number-of-seconds value was not within the supported range.
CEE2EA	3	2506	<JJJJ>, <CCCC> or <CCCCCCCC> was used in a picture string passed to CEEDATM, but the input number-of-seconds value was not within the supported range. The Era could not be determined.
CEE2EM	3	2518	An invalid picture string was specified.
CEE2EV	2	2527	The timestamp string returned by CEEDATM was truncated.
CEE3CF	2	3471	The country code *country-code* was not valid for CEEFMDT. The default date and time picture string *datetime-string* was returned.

Usage notes

- The probable cause for receiving message number 2518 is a picture string that contains a DBCS string that is not valid. You should verify that the data in the picture string is correct.
- To create a null VSTRING, set the length to zero; the content of the text portion does not matter. To create a blank VSTRING, any length greater than zero can be used; the content of the text portion must be spaces or blanks.
- z/OS UNIX consideration—In multithread applications, CEEDATM applies to the enclave.

For more information

- See "CEESECS—Convert timestamp to seconds" on page 405 for more information about the CEESECS callable service.
- See "COUNTRY" on page 22 for more information about the COUNTRY runtime option.

- See "CEEFMDT—Get default date and time format" on page 271 for information about how to get a default timestamp for a given country code.

Examples

1. Following is an example of CEEDATM called by C/C++.

```
/*Module/File Name: EDCDATM   */

#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

             /* September 13, 1991 at 11:23:23 PM */
   _FLOAT8 seconds = 12904183403.0;
   _VSTRING date,date_pic;
   _CHAR80 out_date;
   _FEEDBACK fc;

   strcpy(date_pic.string,
   "Mmmmmmmmmmmmz DD, YYYY at ZH:MI:.SS AP");
   date_pic.length = strlen(date_pic.string);

   CEEDATM(&seconds,&date_pic,out_date,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEDATM failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }

   printf("%.80s\n",out_date);
}
```

2. Following is an example of CEEDATM called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTDATM
      *************************************************
      **                                             **
      ** Function: CEEDATM - convert seconds to      **
      **                     character timestamp     **
      **                                             **
      ** In this example, a call is made to CEEDATM  **
      ** to convert a date represented in Lilian     **
      ** seconds (the number of seconds since        **
      ** 00:00:00 14 October 1582) to a character    **
      ** format (such as 06/02/88 10:23:45). The     **
      ** result is displayed.                        **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDATM.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  DEST          PIC S9(9) BINARY VALUE 2.
       01  SECONDS                 COMP-2.
       01  IN-DATE.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char    PIC X
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                              of IN-DATE.
       01  PICSTR.
           02  Vstring-length      PIC S9(4) BINARY.
```

```
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of PICSTR.
 01  TIMESTP                 PIC X(80).
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.
 PARA-CBLDATM.
*************************************************
** Call CEESECS to convert timestamp of 6/2/88 **
**     at 10:23:45 AM to Lilian representation **
*************************************************
     MOVE 20 TO Vstring-length of IN-DATE.
     MOVE "06/02/88 10:23:45 AM"
         TO Vstring-text of IN-DATE.
     MOVE 20 TO Vstring-length of PICSTR.
     MOVE "MM/DD/YY HH:MI:SS AP"
         TO Vstring-text of PICSTR.
     CALL "CEESECS" USING IN-DATE, PICSTR,
                          SECONDS, FC.

*************************************************
** If CEESECS runs successfully, display result**
*************************************************
     IF  CEE000 of FC  THEN
         DISPLAY Vstring-text of IN-DATE
             " is Lilian second:  " SECONDS
     ELSE
         DISPLAY "CEESECS failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.


*************************************************
** Specify desired format of the output.       **
*************************************************
     MOVE 35 TO Vstring-length OF PICSTR.
     MOVE "ZD Mmmmmmmmmmmmmmz YYYY at HH:MI:SS"
             TO Vstring-text OF PICSTR.
*************************************************
** Call CEEDATM to convert Lilian seconds to   **
**     a character timestamp                   **
*************************************************
     CALL "CEEDATM" USING SECONDS, PICSTR,
                          TIMESTP, FC.

*************************************************
** If CEEDATM runs successfully, display result**
*************************************************
     IF CEE000 of FC  THEN
         DISPLAY "Input seconds of " SECONDS
             " corresponds to: " TIMESTP
```

```
          ELSE
              DISPLAY "CEEDATM failed with msg "
                  Msg-No of FC UPON CONSOLE
              STOP RUN
           END-IF.

           GOBACK.
```

3. Following is an example of CEEDATM called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMDATM

 /****************************************************/
 /**                                                **/
 /** Function: CEEDATM - Convert seconds to         **/
 /**                     character timestamp        **/
 /**                                                **/
 /** In this example, CEEDATM is called to convert  **/
 /** the number of seconds since 00:00:00 14        **/
 /** October 1582 to the character format specified **/
 /** in PICSTR.                                     **/
 /**                                                **/
 /****************************************************/

 PLIDATM: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL SECONDS REAL FLOAT DECIMAL(16);
    DCL PICSTR  CHAR(255) VARYING;
    DCL TIMESTP CHAR(80);
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    SECONDS = 13166064060; /* Input is Lilian seconds*/

    PICSTR = 'ZD Mmmmmmmmmmmmmmz YYYY';  /* Picture  */
          /* string describing desired output format */

    /* Call CEEDATM to convert Lilian seconds to     */
    /*    format specified in PICSTR                 */
    CALL CEEDATM ( SECONDS , PICSTR , TIMESTP , FC );

    /* If CEEDATM ran successfully, print result     */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'Input Lilian seconds correspond to '
          || TIMESTP);
       END;
    ELSE  DO;
       DISPLAY( 'CEEDATM failed with msg '
          || FC.MsgNo );
       STOP;
       END;

  END PLIDATM;
```

Table 27 on page 241 shows the sample output of CEEDATM.

Table 27. Sample output of CEEDATM

input_seconds	picture_string	output_timestamp

12,799,191,601.000	`YYMMDD`
	`HH:MI:SS`
	`YY-MM-DD`
	`YYMMDDHHMISS`
	`YY-MM-DD HH:MI:SS`
	`YYYY-MM-DD HH:MI:SS`
	`AP`
		`880516`
		`19:00:01`
		`88-05-16`
		`880516190001`
		`88-05-16`
		`19:00:01`
		`1988-05-16`
		`07:00:01 PM`

12,799,191,661.986	`DD Mmm YY`
	`DD MMM YY HH:MM`
	`WWW, MMM DD, YYYY`
	`ZH:MI AP`
	`Wwwwwwwwwwz, ZM/ZD/YY`
	`HH:MI:SS.99`
		`16 May 88`
		`16 MAY 88 19:01`
		`MON, MAY 16,`
		`1988 7:01 PM`
		`Monday, 5/16/88`
		`19:01:01.98`

12,799,191,662.009	`YYYY`
	`YY`
	`Y`
	`MM`
	`ZM`
	`RRRR`
	`MMM`
	`Mmm`
	`Mmmmmmmmmm`
	`Mmmmmmmmmz`
	`DD`
	`ZD`
	`DDD`
	`HH`
	`ZH`
	`MI`
	`SS`
	`99`
	`999`
	`AP`
	`WWW`
	`Www`
	`Wwwwwwwwww`
	`Wwwwwwwwwz`
		`1988`
		`88`
		`8`
		`05`
		`5`
		`Vƀƀƀ`
		`MAY`
		`Mayƀƀƀƀƀƀ`
		`May`
		`16`
		`16`
		`137`
		`19`
		`01`
		`02`
		`00`
		`009`
		`PM`
		`MON`
		`Mon`
		`Mondayƀƀƀƀ`
		`Monday`

CEEDAYS—Convert date to Lilian format

CEEDAYS converts a string representing a date into a Lilian format, which represents a date as the number of days from the beginning of the Gregorian calendar. CEEDAYS converts the specified *input_char_date* to a number representing the number of days since day one in the Lilian format: Friday, 14 October, 1582.

The inverse of CEEDAYS is CEEDATE, which converts *output_Lilian_date* from Lilian format to character format.

Do not use CEEDAYS in combination with COBOL intrinsic functions unless the programs are compiled with the INTDATE(LILIAN) compiler option. Use CEECBLDY for COBOL programs that use intrinsic functions and that are compiled with INTDATE(ANSI).

To handle dates earlier than 1601, it is possible to add 4000 to each year, convert to Lilian, calculate, subtract 4000 from the result, and then convert back to character format.

By default, 2-digit years lie within the 100-year range starting 80 years prior to the system date. Thus, in 1995, all 2-digit years represent dates between 1915 and 2014, inclusive. This default range is changed by using the callable service CEESCEN.

Syntax

```
►►──CEEDAYS──(──input_char_date──,──picture_string──,──output_Lilian_date──────►

►──,──fc──)─────────────────────────────────────────────────────────────────────►◄
```

input_char_date **(input)**
: A halfword length-prefixed character string (VSTRING), representing a date or timestamp, in a format conforming to that specified by *picture_string*.

  The character string must contain between 5 and 255 characters, inclusive. *input_char_date* can contain leading or trailing blanks. Parsing for a date begins with the first nonblank character (unless the picture string itself contains leading blanks, in which case CEEDAYS skips exactly that many positions before parsing begins).

  After parsing a valid date, as determined by the format of the date specified in *picture_string*, CEEDAYS ignores all remaining characters. Valid dates range between and include 15 October 1582 to 31 December 9999. See Table 33 on page 487 for a list of valid picture character terms that can be specified in *input_char_date*.

picture_string **(input)**
: A halfword length-prefixed character string (VSTRING), indicating the format of the date specified in *input_char_date*.

  Each character in the *picture_string* corresponds to a character in *input_char_date*. For example, if you specify MMDDYY as the *picture_string*, CEEDAYS reads an *input_char_date* of 060288 as 02 June 1988.

  If delimiters such as a slash (/) appear in the picture string, leading zeros can be omitted. For example, the following calls to CEEDAYS, would each assign the same value, 148155 (02 June 1988), to *lildate*.

```
CALL CEEDAYS('6/2/88'  , 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('06/02/88', 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('060288'  , 'MMDDYY'  , lildate, fc);
CALL CEEDAYS('88154'   , 'YYDDD'   , lildate, fc);
CALL CEEDAYS('1988154' , 'YYYYDDD' , lildate, fc);
```

  Whenever characters such as colons or slashes are included in the *picture_string* (such as `HH:MI:SS YY/MM/DD`), they count as placeholders but are otherwise ignored. See Table 33 on page 487 for a list of valid picture character terms and Table 34 on page 488 for examples of valid picture strings.

  If *picture_string* includes a Japanese Era symbol <JJJJ>, the `YY` position in *input_char_date* is replaced by the year number within the Japanese Era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See

Table 34 on page 488 for an additional example. See also Table 35 on page 488 for a list of Japanese Eras supported by CEEDATE.

If *picture_string* includes era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *input_char_date* is replaced by the year number within the era. See Table 34 on page 488 for an additional example.

output_Lilian_date **(output)**
A 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582. For example, 16 May 1988 is day number 148138.

If *input_char_date* does not contain a valid date, *output_Lilian_date* is set to 0 and CEEDAYS terminates with a non-CEE000 symbolic feedback code.

Date calculations are performed easily on the *output_Lilian_date*, because it is an integer. Leap year and end-of-year anomalies do not affect the calculations.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EB	3	2507	Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.
CEE2EC	3	2508	The date value passed to CEEDAYS or CEESECS was not valid.
CEE2ED	3	2509	Tbe era passed to CEEDAYS or CEESECS was not recognized.
CEE2EH	3	2513	The input date was not within the supported range.
CEE2EL	3	2517	The month value was not recognized.
CEE2EM	3	2518	An incorrect picture string was specified.
CEE2EO	3	2520	CEEDAYS detected non-numeric data in a numeric field, or the date string did not match the picture string.
CEE2EP	3	2521	The year-within-era value passed to CEEDAYS or CEESECS was zero.

Usage notes

- The probable cause for receiving message number 2518 is a picture string that contains a DBCS string that is not valid. You should verify that the data in the picture string is correct.
- z/OS UNIX consideration—In multithread applications, CEEDAYS applies to the enclave.

For more information

- See the INTDATE COBOL compiler installation option in 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) for information about how to get Lilian integer values from COBOL intrinsic functions that are compatible with the Language Environment callable services CEEDAYS and CEEDATE.

- See "CEEDATE—Convert Lilian date to character format" on page 230 for more information about the CEEDATE runtime option.
- See "CEESCEN—Set the century window" on page 394 for more information about the CEESCEN callable service.

Examples

1. Following is an example of CEEDAYS called by C/C++.

```
/*Module/File Name: EDCDAYS   */

#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _INT4 lil_date1,lil_date2;
   _VSTRING date,date_pic;

   /* use CEEDAYS to get the Lilian format */
   strcpy(date.string,"05/14/64");
   date.length = strlen(date.string);
   strcpy(date_pic.string,"MM/DD/YY");
   date_pic.length = strlen(date_pic.string);

   CEEDAYS(&date,&date_pic,&lil_date1,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEDAYS failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }

   /* use CEEDAYS to get the Lilian format */
   strcpy(date.string,"August 14, 1966");
   date.length = strlen(date.string);
   strcpy(date_pic.string,"Mmmmmmmmmmmz DD, YYYY");
   date_pic.length = strlen(date_pic.string);

   CEEDAYS(&date,&date_pic,&lil_date2,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEDAYS failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }

   /* subtract the two Lilian dates to find out */
   /* difference in days */
   printf("The number of days between"
   " May 14, 1964 and August 14, 1966"
   " is: %d\n",lil_date2 - lil_date1);
}
```

2. Following is an example of CEEDAYS called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTDAYS
      *****************************************
      **                                     **
      ** Function: CEEDAYS - convert date to **
      **                     Lilian format   **
      **                                     **
      *****************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDAYS.
```

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  CHRDATE.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                         of CHRDATE.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                         of PICSTR.
 01  LILIAN                  PIC S9(9) BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.
 PARA-CBLDAYS.
*************************************************
** Specify input date and length              **
*************************************************
     MOVE 16 TO Vstring-length of CHRDATE.
     MOVE "1 January 2000"
         TO Vstring-text of CHRDATE.
*************************************************
** Specify a picture string that describes     **
** input date, and the picture string's length.**
*************************************************
     MOVE 25 TO Vstring-length of PICSTR.
     MOVE "ZD Mmmmmmmmmmmmmmz YYYY"
             TO Vstring-text of PICSTR.

*************************************************
** Call CEEDAYS to convert input date to a     **
** Lilian date                                 **
*************************************************
     CALL "CEEDAYS" USING CHRDATE, PICSTR,
                          LILIAN, FC.

*************************************************
** If CEEDAYS runs successfully, display result**
*************************************************
     IF  CEE000 of FC  THEN
         DISPLAY Vstring-text of CHRDATE
             " is Lilian day: " LILIAN
     ELSE
         DISPLAY "CEEDAYS failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEEDAYS called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMDAYS                          */
 /****************************************************/
 /**                                                **/
 /** Function     : CEEDAYS - Convert date to       **/
 /**                          Lilian format         **/
 /**                                                **/
 /** This example converts two dates to the Lilian **/
 /** format in order to calculate the number of    **/
 /** days between them.                             **/
 /**                                                **/
 /****************************************************/

 PLIDAYS: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL CHRDATE  CHAR(255) VARYING;
    DCL CHRD2    CHAR(255) VARYING;
    DCL PICSTR   CHAR(255) VARYING;
    DCL PICST2   CHAR(255) VARYING;
    DCL LILIAN   REAL FIXED BINARY(31,0);
    DCL LIL2     REAL FIXED BINARY(31,0);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);

    /* First date to be converted to Lilian format   */
    CHRDATE = '5/7/69';

    /* Picture string of first input date           */
    PICSTR = 'ZM/ZD/YY';

    /* Call CEEDAYS to convert input date to the     */
    /*    Lilian format                              */
    CALL CEEDAYS ( CHRDATE , PICSTR , LILIAN , FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'The Lilian date for ' || CHRDATE
          || ' is ' || LILIAN );
       END;
    ELSE  DO;
       DISPLAY( 'CEEDAYS failed with msg '
          || FC.MsgNo );
       STOP;
       END;

    /* Second date to be converted to Lilian format  */
    CHRD2 = '1 January 2000';

    /* Picture string of second input date           */
    PICST2 = 'ZD Mmmmmmmmmmmmmmz YYYY';

    /* Call CEEDAYS to convert input date to the     */
    /*    Lilian format                              */
    CALL CEEDAYS ( CHRD2 , PICST2 , LIL2 , FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'The Lilian date for ' || CHRD2
          || ' is ' || LIL2 );
```

```
      END;
   ELSE  DO;
      DISPLAY( 'CEEDAYS failed with msg '
         || FC.MsgNo );
      STOP;
      END;

   /* Subtract the two Lilian dates to find out     */
   /*    the difference in days between the two     */
   /*    input dates                                */
   PUT SKIP LIST( 'The number of days between '
      || CHRDATE || ' and ' || CHRD2 || ' is'
      || LIL2 - LILIAN || '.');

END PLIDAYS;
```

CEEDCOD—Decompose a condition token

CEEDCOD alters an existing condition token. Language Environment-conforming HLLs can decompose or alter the condition token fields without using the CEEDCOD service. See the CEESGL HLL examples in "Examples" on page 417 for examples of how to alter the condition token field.

Syntax

```
►►──CEEDCOD──(──cond_token──,──c_1──,──c_2──,──case──,──severity──,──────────►

►─control──,──facility_ID──,──i_s_info──,──fc──)─────────────────────────────►◄
```

cond_token **(input)**
: A 12-byte condition token representing the current condition or feedback information.

c_1 **(output)**
: A 2-byte binary integer representing the value of the first 2 bytes of the condition_ID.

c_2 **(output)**
: A 2-byte binary integer representing the value of the second 2 bytes of the condition_ID. See "CEENCOD—Construct a condition token" on page 377 for a detailed explanation of the condition_ID.

case **(output)**
: A 2-byte binary integer field defining the format of the condition_ID portion of the token. A value of 1 identifies a case 1 condition. A value of 2 identifies a case 2 condition. The values 0 and 3 are reserved.

severity **(output)**
: A 2-byte binary integer representing the severity of the condition. *severity* specifies the following values:

0	Information only (or, if the entire token is zero, no information).
1	Warning—service completed, probably correctly.
2	Error detected—correction attempted; service completed, perhaps incorrectly.
3	Severe error—service not completed.

4 Critical error—service not completed; condition signaled. A critical error is a condition that jeopardizes the environment. If a critical error occurs during a Language Environment callable service, instead of returning synchronously to the caller, the condition manager is always signaled.

control **(output)**
A 2-byte binary integer containing flags describing aspects of the state of the condition. Valid values for the control field are 1 and 0.

1 Indicates that the *facility_ID* is assigned by IBM.
0 indicates the *facility_ID* is assigned by the user.

facility_ID **(output)**
A 3-character field containing three alphanumeric characters identifying the product generating the condition or feedback information.

i_s_info
A fullword binary integer that identifies the ISI associated with the given instance of the condition represented by the condition token where it is contained. If an ISI is not associated with a given condition token, the *i_s_info* field contains a value of binary zero.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE036	3	0102	An unrecognized condition token was passed to *routine* and could not be used.

Usage notes

- C/C++ considerations—The structure of the condition token (type_FEEDBACK) is described in the `leawi.h` header file shipped with Language Environment. C users can assign values directly to the fields of the token in the header file without using the CEENCOD service. The layout of the type_FEEDBACK condition token in the header file is shown in Figure 12 on page 379.
- z/OS UNIX consideration—In multithread applications, CEEDCOD affects only the calling thread.

For more information

- See the CEESGL HLL examples starting in "Examples" on page 417 for examples of how to alter the condition token field.
- See "CEENCOD—Construct a condition token" on page 377 for a detailed explanation of the condition_ID.
- See CEENCOD "Usage notes" on page 378 for a discussion of case 1 and case 2 types.
- See the *facility_ID* parameter of "CEENCOD—Construct a condition token" on page 377 for more information.
- For more C user information about assigning values directly to the fields of the token in the header file without using the CEENCOD service, see the example for "CEESGL—Signal a condition" on page 415.

Examples

1. Following shows an example of CEEDCOD being called by C/C++.

```
/*Module/File Name: EDCDCOD   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

/****************************************************/
/* In C/C++ it is not necessary to use this service.*/
/* The fields can be manipulated directly.  See the */
/* example for CEESGL to see how to manipulate      */
/* condition token fields directly.                 */
/****************************************************/

int main(void) {

  _FEEDBACK fc,newfc;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi, heapid, size;
  _POINTER address;

  heapid = 0;
  size = 4000;

  CEEGTST(&heapid,&size,&address,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEGTST failed with msgno %d\n",
       fc.tok_msgno);
     exit(2999);
  }

  /* decompose the feedback token to check for errors */
  CEEDCOD(&fc,&c_1,&c_2,&cond_case,&sev,&control,facid,;
          &isi,&newfc);

  if ( _FBCHECK ( newfc , CEE000 ) != 0 ) {
     printf("CEEDCOD failed with msgno %d\n",
       newfc.tok_msgno);
     exit(2889);
  }
  if (c_1 != 0 || c_2 != 0)
     printf(
     "c_1 and c_2 returned from CEEDCOD should be 0\n");
 /*

⋮
 */
}
```

2. Following an example of CEEDCOD being called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTDCOD
      ***********************************************
      **                                           **
      ** Function: CEEDCOD - Decompose a condition **
       **                     token                **
      **                                           **
      ** In this example, a call is made to        **
      ** CEEGTST in order to obtain a condition    **
      ** token to use in the call to CEEDCOD.      **
      ** A call could also have been made to any   **
      ** other Lang Env svc., or a condition token **
** could have been constructed using         **
** CEEDCOD.                                  **
**                                           **
***********************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLDCOD.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  HEAPID                  PIC S9(9) BINARY.
 01  HPSIZE                  PIC S9(9) BINARY.
 01  ADDRSS                  USAGE POINTER.
 01  SEV                     PIC S9(4) BINARY.
 01  MSGNO                   PIC S9(4) BINARY.
 01  CASE                    PIC S9(4) BINARY.
 01  SEV2                    PIC S9(4) BINARY.
 01  CNTRL                   PIC S9(4) BINARY.
 01  FACID                   PIC X(3).
 01  ISINFO                  PIC S9(9) BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 01  FC2.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.
*************************************************
** Call any Lang Env svc to receive a condition**
**     token to use as input to CEEDCOD.      **
*************************************************

PARA-CBLGTST.
************************************************
** Specify 0 to get storage from the initial  **
**     heap.                                  **
** Specify 4000 to get 4000 bytes of storage. **
** Call CEEGTST to obtain storage.            **
************************************************

     MOVE 0 TO HEAPID.
     MOVE 4000 TO HPSIZE.

     CALL "CEEGTST" USING HEAPID, HPSIZE,
                          ADDRSS, FC.

 PARA-CBLDCOD.
************************************************
```

```
** Use the FC returned from CEEGTST as an     **
**     input condition token to CEEDCOD.      **
************************************************

     CALL "CEEDCOD" USING FC, SEV, MSGNO, CASE,
                          SEV2, CNTRL, FACID,
                          ISINFO, FC2.
     IF CEE000 of FC2  THEN
         DISPLAY "CEEGTST completed with msg "
             MSGNO ", Severity " SEV ", Case "
             CASE ", Control " CNTRL ", and "
             "Instance-Specific Information of "
             ISINFO "."
     ELSE
         DISPLAY "CEEDCOD failed with msg "
             Msg-No of FC2 UPON CONSOLE
     END-IF.
     GOBACK.
```

3. Following is an example of CEEDCOD called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMDCOD                          */
 /****************************************************/
 /**                                                **/
 /** Function: CEEDCOD - decompose a condition     **/
 /**                    token                      **/
 /**                                                **/
 /** In this example, a call is made to CEEGTST to **/
 /** receive a condition token to decompose.       **/
 /** A call could have been made to any LE/370     **/
 /** service. The condition token returned by      **/
 /** CEEGTST is used as input to CEEDCOD.          **/
 /**                                                **/
 /****************************************************/
 PLIDCOD: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL HEAPID  REAL FIXED BINARY(31,0);
    DCL STGSIZE REAL FIXED BINARY(31,0);
    DCL ADDRSS  POINTER;
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    DCL SEV     REAL FIXED BINARY(15,0);
    DCL MSGNO   REAL FIXED BINARY(15,0);
    DCL CASE    REAL FIXED BINARY(15,0);
    DCL SEV2    REAL FIXED BINARY(15,0);
    DCL CNTRL   REAL FIXED BINARY(15,0);
    DCL FACID   CHARACTER ( 3 );
    DCL ISINFO  REAL FIXED BINARY(31,0);
    DCL 01 FC2,                   /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
```

```
          03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);

   HEAPID = -1;    /* invalid heap ID              */
   STGSIZE = 4000; /* request 4000 bytes of storage */

   /* Call any service (in this case, CEEGTST) to   */
   /*       create a condition token to decompose   */
   CALL CEEGTST ( HEAPID , STGSIZE , ADDRSS , FC );
   /* Call CEEDCOD with the condition token        */
   /*    returned in FC from CEEGTST               */
   CALL CEEDCOD ( FC , SEV , MSGNO , CASE , SEV2 ,
         CNTRL , FACID , ISINFO , FC2 );
   IF  FBCHECK( FC2, CEE000)  THEN  DO;
      PUT SKIP LIST( 'Feedback token from CEEGTST has'
         || ' Severity of ' || SEV
         || ', Message number of ' || MSGNO
         || ', Case of ' || CASE || ',' );
      PUT SKIP LIST( '   Severity 2 of ' || SEV2
         || ', Control of ' || CNTRL
         || ', Facility ID of ' || FACID
         || ', and I-S-Info of ' || ISINFO || '.' );
      END;
   ELSE  DO;
      DISPLAY( 'CEEDCOD failed with msg '
         || FC2.MsgNo );
      STOP;
      END;
END PLIDCOD;
```

CEEDLYM—Suspend processing of the active enclave in milliseconds

CEEDLYM provides a service for Language Environment-conforming applications that suspends the processing of the active enclave for a specified number of milliseconds. The maximum is 1 hour.

Syntax

```
►►──CEEDLYM──(──input_milliseconds──,──fc──)─────────────────────────────►◄
```

input_milliseconds
: A full-word binary value in the range of 0 to 3,600,000 that specifies the total number of milliseconds during which the enclave should be suspended.

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE3QQ	1	CEE3930W	The input value *input_value* in a call to the callable service *callable_service_name* was not within the valid range.
CEE3QR	1	CEE3931W	CEEDLYM was invoked in a CICS environment.

Code	Severity	Message number	Message text
CEE3QS	1	CEE3932W	The system service *system_service* failed with return code *return_code* and reason code *reason_code*.

Usage notes

- CICS consideration—CEEDLYM is not available under CICS.
- z/OS UNIX consideration—CEEDLYM is handled by the z/OS UNIX System Services when POSIX is set to ON.
- This service is not intended for timing requests. Delays up to the nearest millisecond might occur in some circumstances.
- In a multi-threaded application, only the calling thread is suspended.

Examples

1. Following is an example of CEEDLYM called by C/C++.

```
/*Module/File Name: EDCDLYM  */
/*********************************************************************/
/*                                                                   */
/* THIS EXAMPLE CALLS CEEDLYM TO SUSPEND PROCESSING OF THE ACTIVE    */
/* ENCLAVE FOR SPECIFIED NUMBER OF MILLISECONDS.                     */
/*                                                                   */
/*********************************************************************/
#include <string.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void)
{
  _INT4 millisecs, lil_date;
  _FLOAT8   local_date;
  _CHAR17   gregorian_date;
  _FEEDBACK fc1, fc2;

  /* Get current date and time */
  CEELOCT(&lil_date,&local_date,gregorian_date,&fc1);
  if ( _FBCHECK ( fc1 , CEE000 ) != 0 ) {
    printf("CEELOCT failed with message number %d\n",
           fc1.tok_msgno);
    exit(2999);
  }
  printf("CEEDLYM Start  time :  %.17s\n", gregorian_date);

  millisecs = 5000;
  CEEDLYM(&millisecs,&fc2);

  if ( _FBCHECK ( fc2 , CEE000 ) != 0 ) {
    printf("CEEDLYM failed with message number %d\n",fc2.tok_msgno);
    exit(999);
  }

  CEELOCT(&lil_date,&local_date,gregorian_date,&fc1);
  if ( _FBCHECK ( fc1 , CEE000 ) != 0 ) {
    printf("CEELOCT failed with message number %d\n",
           fc1.tok_msgno);
    exit(2999);
  }
  printf("CEEDLYM Finish time :  %.17s\n",gregorian_date);
}
```

2. Following is an example of CEEDLYM called by COBOL.

```
CBL LIB,QUOTE
      *************************************************
      *MODULE/FILE NAME: IGZTDLYM
      *****************************************************************
      **                                                             **
      ** FUNCTION: CEEDLYM - SUSPEND ENCLAVE EXECUTION IN MILLISECS  **
      **                                                             **
      ** THIS EXAMPLE CALLS CEEDLYM TO SUSPEND PROCESSING OF THE     **
      ** ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF MILLISECONDS.        **
      **                                                             **
      *****************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDLYM.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  LILIAN                 PIC S9(9) BINARY.
       01  SECONDS                COMP-2.
       01  GREGORN                PIC X(17).
       01  MILLISECS              PIC S9(9) BINARY.
       01  FC1.
           02  CONDITION-TOKEN-VALUE.
           COPY  CEEIGZCT.
               03  CASE-1-CONDITION-ID.
                  04  SEVERITY     PIC S9(4) BINARY.
                  04  MSG-NO       PIC S9(4) BINARY.
               03  CASE-2-CONDITION-ID
                        REDEFINES CASE-1-CONDITION-ID.
                  04  CLASS-CODE   PIC S9(4) BINARY.
                  04  CAUSE-CODE   PIC S9(4) BINARY.
               03  CASE-SEV-CTL    PIC X.
               03  FACILITY-ID     PIC XXX.
           02  I-S-INFO           PIC S9(9) BINARY.
       01  FC2.
           02  CONDITION-TOKEN-VALUE.
           COPY  CEEIGZCT.
               03  CASE-1-CONDITION-ID.
                  04  SEVERITY     PIC S9(4) BINARY.
                  04  MSG-NO       PIC S9(4) BINARY.
               03  CASE-2-CONDITION-ID
                        REDEFINES CASE-1-CONDITION-ID.
                  04  CLASS-CODE   PIC S9(4) BINARY.
                  04  CAUSE-CODE   PIC S9(4) BINARY.
               03  CASE-SEV-CTL    PIC X.
               03  FACILITY-ID     PIC XXX.
           02  I-S-INFO           PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLDLYM.
           DISPLAY "CEEDLYM - BEGINS"
           CALL "CEELOCT" USING LILIAN, SECONDS, GREGORN, FC1.
           IF CEE000 OF FC1 THEN
             DISPLAY "START  DATE & TIME: " GREGORN
           ELSE
             DISPLAY "CEELOCT FAILED WITH MSG "
                      MSG-NO OF FC1
             STOP RUN
           END-IF.
           MOVE 5000 TO MILLISECS.
           CALL "CEEDLYM" USING MILLISECS, FC2.
           IF NOT CEE000 OF FC2 THEN
              DISPLAY "CEEDLYM FAILED WITH MSG "
                       MSG-NO OF FC2
              STOP RUN
           END-IF.
           DISPLAY "CEEDLYM - COMPLETED"
           CALL "CEELOCT" USING LILIAN, SECONDS,
                          GREGORN, FC1.
```

```
IF CEE000 OF FC1 THEN
  DISPLAY "FINISH DATE & TIME: " GREGORN
ELSE
  DISPLAY "CEELOCT FAILED WITH MSG "
           MSG-NO OF FC1
  STOP RUN
END-IF.
GOBACK.
```

3. Following is an example of CEEDLYM called by PL/I.

```
*PROCESS MACRO;
  /*MODULE/FILE NAME: IBMDLYM                                       */
  /******************************************************************/
  /**                                                              **/
  /** FUNCTION: CEEDLYM - SUSPENDS ENCLAVE EXECUTION IN MILLISECS   **/
  /**                                                              **/
  /** THIS EXAMPLE CALLS CEEDLYM TO SUSPEND PROCESSING OF THE       **/
  /** ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF MILLISECS.             **/
  /**                                                              **/
  /******************************************************************/
  PLIDLYM: PROCEDURE OPTIONS (MAIN) REORDER;

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DECLARE MILLISECS REAL FIXED BINARY(31,0);
    DECLARE LILIAN    REAL FIXED BINARY(31,0);
    DECLARE SECONDS   REAL FLOAT DECIMAL(16);
    DECLARE GREGORN   CHARACTER (17);

    DECLARE 01 FC1,              /* FEEDBACK TOKEN FOR CEELOCT */
               03 MSGSEV       REAL FIXED BINARY(15,0),
               03 MSGNO        REAL FIXED BINARY(15,0),
               03 FLAGS,
                  05 CASE      BIT(2),
                  05 SEVERITY  BIT(3),
                  05 CONTROL   BIT(3),
               03 FACID        CHAR(3),
               03 ISI          REAL FIXED BINARY(31,0);

    DECLARE 01 FC2,             /* FEEDBACK TOKEN FOR CEEDLYM */
               03 MSGSEV       REAL FIXED BINARY(15,0),
               03 MSGNO        REAL FIXED BINARY(15,0),
               03 FLAGS,
                  05 CASE      BIT(2),
                  05 SEVERITY  BIT(3),
                  05 CONTROL   BIT(3),
               03 FACID        CHAR(3),
               03 ISI          REAL FIXED BINARY(31,0);

    CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC1 );
    IF FBCHECK(FC1, CEE000)  THEN  DO;
      PUT SKIP LIST ( 'CEEDLYM START  DATE AND TIME: ' || GREGORN );
    END;
    ELSE DO;
      PUT ( 'CEELOCT FAILED WITH MSG ' || FC1.MSGNO );
      STOP;
    END;

    MILLISECS = 6000;
    CALL CEEDLYM(MILLISECS, FC2);
    IF FBCHECK(FC2, CEE000) THEN DO;
      CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC1 );
      IF  FBCHECK(FC1, CEE000)  THEN  DO;
        PUT SKIP LIST ( 'CEEDLYM FINISH DATE AND TIME: ' || GREGORN );
      END;
        ELSE DO;
```

```
          PUT ('CEELOCT FAILED WITH MSG ' || FC1.MSGNO );
          STOP;
        END;
        PUT SKIP LIST ( 'CEEDLYM IS SUCCESSFUL!' );
      END;
      ELSE DO;
        PUT SKIP LIST ( 'CEEDLYM FAILED WITH MSG ' || FC2.MSGNO );
        STOP;
      END;

    END PLIDLYM;
```

CEEDSHP—Discard heap

CEEDSHP discards an entire heap created by CEECRHP or by CEEGTST. CEECRHP and CEEGTST return a unique *heap_id* to the caller; use this ID in the CEEDSHP call. A *heap_id* of 0 is not permitted with CEEDSHP.

Discarding a heap with CEEDSHP immediately returns all storage allocated to the heap to the operating system, even if the KEEP suboption has been specified with the HEAP runtime option.

Syntax

```
►►──CEEDSHP──(──heap_id──,──fc──)──────────────────────────────────────►◄
```

heap_id **(input)**
: A fullword binary signed integer. *heap_id* is a token specifying the discarded heap. A *heap_id* of 0 is not valid; the initial heap is logically created during enclave initialization and cannot be discarded.

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P3	3	0803	The heap identifier in a get storage request or a discard heap request was unrecognized.
CEE0PC	3	0812	An invalid attempt to discard the Initial Heap was made.

Usage notes

- After the call to CEEDSHP, any existing pointers to storage allocated from this heap are dangling pointers, that is, pointers to storage that is freed. Using these pointers can cause unpredictable results.
- z/OS UNIX considerations—CEEDSHP applies to the enclave. Language Environment frees all storage in the heap regardless of which thread allocated it.

For more information

- For more information about the CEEDSHP callable service, see "CEECRHP—Create new additional heap" on page 221.
- For more information about the CEEGTST callable service, and "CEEGTST—Get heap storage" on page 313.

Examples

1. Following is an example of CEEDSHP being called by C/C++.

```
/*Module/File Name: EDCDSHP   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _INT4 heapid, size, increment, options;
   _FEEDBACK fc;
 /*  .
     .
     .  */
   heapid = 0;        /* heap identifier is set */
                      /* by CEECRHP */
   size = 4096;       /* initial size of heap */
                      /* (in bytes)       */
   increment = 4096;  /* increment to extend */
                      /* the heap by */
   options = 72;      /* set up heap as */
                      /* (,,ANYWHERE,FREE)*/

   /* create heap using CEECRHP */
   CEECRHP(&heapid,&size,&increment,&options,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEECRHP failed with message number %d\n",
             fc.tok_msgno);
      exit(99);
   }
 /*  .
     .
     .  */

   /* discard the heap that was previously created */
   /* using CEECRHP */
   CEEDSHP(&heapid,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEDSHP failed with message number %d\n",
             fc.tok_msgno);
      exit(99);
   }
 /*  .
     .
     .  */
}
```

2. Following an example of CEEDSHP being called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTDSHP
      *************************************************
      **                                             **
      ** Function: CEEDSHP - discard heap            **
      **                                             **
      ** In this example, a new additional heap is   **
      ** created a call to CEECRHP, and then         **
      ** discarded through a call to CEEDSHP.        **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDSHP.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  HEAPID                  PIC S9(9) BINARY.
       01  HPSIZE                  PIC S9(9) BINARY.
       01  INCR                    PIC S9(9) BINARY.
       01  OPTS                    PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLCRHP.
      ************************************************
      ** Specify 0 for HEAPID, and heap id will      **
      **     be set by CEECRHP.                      **
      ** Heap size and increment will each be 4096   **
      **     bytes.                                  **
      ** Specify 00 for OPTS, and HEAP attributes    **
      **     will be inherited from the initial heap **
      **     (copied from the HEAP runtime option). **
      ************************************************
           MOVE 0 TO HEAPID.
           MOVE 4096 TO HPSIZE.
           MOVE 4096 TO INCR.
           MOVE 00 TO OPTS.

           CALL "CEECRHP" USING HEAPID, HPSIZE,
                                INCR, OPTS, FC.
           IF CEE000 of FC  THEN
               DISPLAY "Created heap number " HEAPID
                    " which is " HPSIZE " bytes long"
           ELSE
               DISPLAY "CEECRHP failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
      ************************************************
      ** To discard the heap, call CEEDSHP with the  **
      **     heap id returned from CEECRHP.          **
      ************************************************
           CALL "CEEDSHP" USING HEAPID, FC.
           IF CEE000 of FC THEN
               DISPLAY "Disposed of heap #  " HEAPID
           ELSE
               DISPLAY "CEEDSHP failed with msg "
```

```
                Msg-No of FC UPON CONSOLE
        END-IF.

        GOBACK.
```

3. Following is an example of CEEDSHP being called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMDSHP                        */
 /****************************************************/
 /**                                              **/
 /** Function: CEEDSHP - discard heap             **/
 /**                                              **/
 /** In this example, calls are made to CEECRHP    **/
 /** and CEEDSHP to create a heap of 4096 bytes    **/
 /** and then discard it.                          **/
 /**                                              **/
 /****************************************************/

 PLIDSHP: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL HEAPID REAL FIXED BINARY(31,0) ;
    DCL HPSIZE REAL FIXED BINARY(31,0) ;
    DCL INCR REAL FIXED BINARY(31,0) ;
    DCL OPTS REAL FIXED BINARY(31,0) ;
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    DCL 01 FC2,                   /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    HEAPID = 0;    /* HEAPID will be set and         */
                   /*    returned by CEECRHP         */
    HPSIZE = 4096; /* Initial size of heap in bytes  */
    INCR = 4096;   /* Number of bytes to extend      */
                   /*    heap by                     */
    OPTS = 00;     /* Set up heap with the same      */
                   /*    attributes as the initial   */
                   /*    heap (HEAPID = 0)           */

    /* Call CEECRHP to set up new heap               */
    CALL CEECRHP ( HEAPID, HPSIZE, INCR,
          OPTS, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'Created heap number ' || HEAPID
          || ' consisting of ' || HPSIZE || ' bytes' );
       END;
    ELSE  DO;
       DISPLAY( 'CEECRHP failed with msg '
          || FC.MsgNo );
```

```
      STOP;
      END;

   /* Call CEEDSHP to discard heap with the id     */
   /*    returned by CEECRHP                       */
   CALL CEEDSHP ( HEAPID, FC2 );
   IF  FBCHECK( FC2, CEE000)  THEN  DO;
      PUT SKIP LIST( 'Disposed of heap number '
         || HEAPID );
      END;
   ELSE  DO;
      DISPLAY( 'CEEDSHP failed with msg '
         || FC2.MsgNo );
      STOP;
      END;


END PLIDSHP;
```

CEEDYWK—Calculate day of week from Lilian date

CEEDYWK calculates the day of the week on which a Lilian date falls. The day of the week is returned to the calling routine as a number between 1 and 7. The number returned by CEEDYWK is useful for end-of-week calculations.

Syntax

```
►►──CEEDYWK──(──input_Lilian_date──,──output_day_no──,──fc──)──────────────►◄
```

input_Lilian_date **(input)**
A 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582. For example, 16 May 1988 is day number 148138. The valid range of *input_Lilian_date* is between 1 and 3,074,324 (15 October 1582 and 31 December 9999).

output_day_no **(output)**
A 32-bit binary integer representing *input_Lilian_date*'s day-of-week: 1 equals Sunday, 2 equals Monday, ..., 7 equals Saturday. If *input_Lilian_date* is invalid, *output_day_no* is set to 0 and CEEDYWK terminates with a non-CEE000 symbolic feedback code.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EG	3	2512	The Lilian date value passed in a call to CEEDATE or CEEDYWK was not within the supported range.

Usage notes

- z/OS UNIX consideration—In multithread applications, CEEDYWK affects only the calling thread.
- COBOL consideration—The CEEDYWK callable service has different values than the COBOL ACCEPT statement with conceptual data item DAY-OF-WEEK. Use one or the other, not both.

Examples

1. Following is an example of CEEDYWK called by C/C++.

```
/*Module/File Name: EDCDYWK   */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main (void) {

  _INT4 in_date, day;
  _FEEDBACK fc;

  in_date = 139370;  /* Thursday */

  CEEDYWK(&in_date,&day,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEDYWK failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  printf("Lilian date %d, occurs on a ",in_date);
  switch(day) {
     case 1: printf("Sunday.\n");
             break;
     case 2: printf("Monday.\n");
             break;
     case 3: printf("Tuesday.\n");
             break;
     case 4: printf("Wednesday.\n");
             break;
     case 5: printf("Thursday.\n");
             break;
     case 6: printf("Friday.\n");
             break;
     case 7: printf("Saturday.\n");
             break;
     default: printf(
            " ERROR! DAY RETURN BY CEEDYWK UNKNOWN\n");
             break;
  }
}
```

2. Following is an example of CEEDYWK called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTDYWK
      *************************************************
      **                                             **
      ** CBLDYWK - Call CEEDYWK to calculate the     **
      **           day of the week from Lilian date **
      **                                             **
      ** In this example, a call is made to CEEDYWK **
      ** to return the day of the week on which a    **
      ** Lilian date falls. (A Lilian date is the    **
      ** number of days since 14 October 1582)      **
**                                              **
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLDYWK.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  LILIAN                 PIC S9(9) BINARY.
 01  DAYNUM                 PIC S9(9) BINARY.
 01  IN-DATE.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char        PIC X,
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of IN-DATE.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char        PIC X,
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of PICSTR.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info           PIC S9(9) BINARY.

 PROCEDURE DIVISION.
 PARA-CBLDAYS.
** Call CEEDAYS to convert date of 6/2/88 to
**     Lilian representation
     MOVE 6 TO Vstring-length of IN-DATE.
     MOVE "6/2/88" TO Vstring-text of IN-DATE(1:6).
     MOVE 8 TO Vstring-length of PICSTR.
     MOVE "MM/DD/YY" TO Vstring-text of PICSTR(1:8).
     CALL "CEEDAYS" USING IN-DATE, PICSTR,
         LILIAN, FC.

** If CEEDAYS runs successfully, display result.
     IF  CEE000 of FC  THEN
         DISPLAY Vstring-text of IN-DATE
             " is Lilian day: " LILIAN
     ELSE
         DISPLAY "CEEDAYS failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
 PARA-CBLDYWK.

** Call CEEDYWK to return the day of the week on
** which the Lilian date falls
     CALL "CEEDYWK" USING LILIAN , DAYNUM , FC.

** If CEEDYWK runs successfully, print results
     IF CEE000 of FC  THEN
         DISPLAY "Lilian day " LILIAN
             " falls on day " DAYNUM
             " of the week, which is a:"
```

```
** Select DAYNUM to display the name of the day
**     of the week.
           EVALUATE DAYNUM
             WHEN 1
               DISPLAY "Sunday."
             WHEN 2
               DISPLAY "Monday."
             WHEN 3
               DISPLAY "Tuesday"
             WHEN 4
               DISPLAY "Wednesday."
             WHEN 5
               DISPLAY "Thursday."
             WHEN 6
               DISPLAY "Friday."
             WHEN 7
               DISPLAY "Saturday."
           END-EVALUATE
       ELSE
           DISPLAY "CEEDYWK failed with msg "
               Msg-No of FC UPON CONSOLE
           STOP RUN
       END-IF.

       GOBACK.
```

3. Following is an example of CEEDYWK called by PL/I.

CEEENV—Process environmental variables

CEEENV processes environment variables depending on the function code passed in as input. Based on the input to this function, CEEENV can do the following:

- Obtain the value for an existing environment variable
- Create a new environment variable with a value
- Clear all environment variables
- Delete an existing environment variable
- Overwrite the value for an existing environment variable

Syntax

```
►►──CEEENV──(──function_code──,──name_length──,──name──,──value_length──,────────►

►─value──,──fc──)────────────────────────────────────────────────────────────►◄
```

function_code **(input)**
A fullword binary integer containing the function code of one of the following values:

1	Searches the environment table for environment variables specified by *name* and if found returns a pointer to *value*.
2	Adds an environment variable to the environment table. It does not overwrite an existing environment variable.
3	Clears all environment variables from the environment table.
4	Deletes an environment variable from the environment table.

5 Overwrites an existing environment variable in the environment table and adds the environment variable if it does not exist.

name_length **(input)**
A fullword binary integer containing the length of the name for the environment variable. If the request is for function code 3, this argument is ignored.

name **(input)**
Specifies the name of an environment variable. If the request is for function code 3, this argument is ignored.

value_length **(input/output)**
A fullword binary integer containing the length of the value for the environment variable. This argument is input for setting or modifying an environment variable and is output for getting an environment variable value. If the request is for function code 3 or 4, this argument is ignored.

value **(input/output)**
A field that contains the address of a string that contains the value of the environment variable. This argument is input for setting or modifying an environment variable and is output for getting an environment variable value. If the request is for function code 3 or 4, this argument is ignored. For function code 1 (get), the value address is 0 (NULL) if the name is not found in the environment.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE51O	3	5176	Not enough memory available.
CEE51P	3	5177	Bad input character detected for name or value.
CEE51Q	3	5178	Bad address detected for the ENVAR anchor or environment variable array.
CEE51R	3	5179	A parameter to the environment variable processing routine contained a value that was not valid.
CEE51S	0	5180	The specified environment variable name already exists.

Usage notes

- The environment array is searched sequentially and the first occurrence of *name* is used.
- Because an application can manipulate the environment using the environ pointer, Language Environment cannot guarantee a single instance of any particular environment variable.
- For a function code 1 request, the storage returned for the value character string is supplied by Language Environment. There is one buffer per thread, making it the user's responsibility to use or save the value before the next call for a function code 1 on that thread.
- Specifying 0 for *value_length* with function code 2 or 5 results in the environment variable being removed from the environment.

- Applications should not define environment variables that begin with the characters "_BPXK_", "_EDC_", or "_CEE_" because there might be conflicts with variable names reserved for z/OS that begin with those characters.
- *name* and *value* are copied into storage owned by Language Environment.
- If a function code 1 request is made and the variable *name* is not found in the environment, *value_length* is set to 0 upon return.
- A NULL character in *name* is not valid and causes feedback code CEE51P to be returned.
- A NULL character in *value* is interpreted as a string terminator. If a NULL character is imbedded, CEEENV will truncate the *value* string at the last character preceding the NULL.

Examples

1. Following is an example of CEEENV called by C/C++.

```
/*Module/File Name: EDCENV    */
/*******************************************************************/
/*                                                                 */
/* THIS EXAMPLE  CALLS CEEENV TO CLEAR THE ENVIRONMENT VARIABLE    */
/* ARRAY, SET AN ENVIRONMENT VARIABLE AND THEN GET THE VARIABLE    */
/* FROM THE ARRAY.                                                 */
/*                                                                 */
/*******************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>

int main(void) {
  _INT4 func_code;
  _INT4 name_len;
  _POINTER name = (_POINTER)malloc(255);
  _INT4 value_len;
  _POINTER value_ptr;
  _FEEDBACK fc;
  char value[255];

  /* Clearing all the environment variables */
  func_code = 3;
  CEEENV(func_code,name_len,name,value_len,value_ptr,fc);

  /* Setting a new environment variable */
  func_code = 2;
  strcpy(name,"ENVAR1");
  strcpy(value,"DEFAULT");
  name_len = strlen(name);
  value_len = strlen(value);
  value_ptr = value;
  CEEENV(func_code,name_len,name,value_len,value_ptr,fc);

  /* Getting the value of the new variable */
  func_code = 1;
  strcpy(name,"ENVAR1");
  strcpy(value,"");
  value_ptr = value;
  CEEENV(func_code,name_len,name,value_len,value_ptr,fc);

  if (value_len != 0)
    printf("$%s=%s\n",name,value_ptr);
  else
    printf("$%s not found\n",name);

}
```

2. Following is an example of CEEENV called by COBOL.

```
CBL LIB,QUOTE
      *****************************************************************
      *Module/File Name: IGZTENV
      *****************************************************************
      ** Function: CEEENV - Process environment variables            *
      **                                                              *
      ** This example calls CEEENV to clear the environment          *
      ** variable array, set an environment variable and then get    *
      ** the variable from the array.                                 *
      **                                                              *
      *****************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTENV.

       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  FUNCCODE  PIC 9(9) BINARY.
       01  NAMELEN   PIC 9(9) BINARY.
       01  VALUELEN  PIC 9(9) BINARY.
       01  NAME      PIC X(255).
       01  VALPTR    POINTER.

       01  FC.
           02  CONDITION-TOKEN-VALUE.
           COPY  CEEIGZCT.
               03  CASE-1-CONDITION-ID.
                  04  SEVERITY     PIC S9(4) BINARY.
                  04  MSG-NO       PIC S9(4) BINARY.
               03  CASE-SEV-CTL    PIC X.
               03  FACILITY-ID     PIC XXX.
           02  I-S-INFO            PIC S9(9) BINARY.
       01  VAL       PIC X(255).

       LINKAGE SECTION.
       01  VAR       PIC X(5000).

       PROCEDURE DIVISION.
       MAIN-PROG.

      *************************************************
      **   Clear environment array
      *************************************************
           MOVE 3 TO FUNCCODE.
           CALL "CEEENV" USING FUNCCODE,
                               NAMELEN,
                               NAME,
                               VALUELEN,
                               VALPTR,
                               FC.
      *************************************************
      **   Set an environment variable
      *************************************************
           MOVE 2 TO FUNCCODE.
           MOVE "ENVAR1" TO NAME.
           MOVE "DEFAULT" TO VAL.
           MOVE 6 TO NAMELEN.
           MOVE 7 TO VALUELEN.
           SET VALPTR TO ADDRESS OF VAL.
           CALL "CEEENV" USING FUNCCODE,
                             NAMELEN,
                             NAME,
                             VALUELEN,
                             VALPTR,
                             FC.

      *************************************************
```

```
     **   Get the environment variable
     **************************************************
          MOVE 1 TO FUNCCODE.
          MOVE " " TO VAL.
          MOVE 0 TO VALUELEN.
          CALL "CEEENV" USING FUNCCODE,
                              NAMELEN,
                              NAME,
                              VALUELEN,
                              VALPTR,
                              FC.

          IF VALUELEN NOT = 0 THEN
              SET ADDRESS OF VAR TO VALPTR
              DISPLAY NAME(1:NAMELEN) "=" VAR(1:VALUELEN)
          ELSE
              DISPLAY NAME " NOT FOUND"
          END-IF.

          GOBACK.
```

3. Following is an example of CEEENV called by PL/I.

```
*PROCESS MACRO;
  /*****************************************************/
  /*Module/File Name: IBMENV                                       */
  /*****************************************************************/
  /**                                                            **/
  /** Function: CEEENV - process environment variables           **/
  /**                                                            **/
  /** This example calls CEEENV to clear the environment variable  **/
  /** array, set an environment variable and then get the variable  **/
  /** from the array.                                             **/
  /**                                                            **/
  /*****************************************************************/
  PLIENV: PROCEDURE OPTIONS (MAIN) REORDER;

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DECLARE FUNC_CODE       REAL FIXED BINARY(31,0);
    DECLARE NAME_LEN        REAL FIXED BINARY(31,0);
    DECLARE VALUE_LEN       REAL FIXED BINARY(31,0);
    DECLARE NAME            CHAR(255) INIT('');
    DECLARE VALUE           CHAR(255) INIT('');
    DECLARE VALUE2          CHAR(255) BASED(VALUE_PTR);
    DECLARE VALUE_PTR       POINTER;

    DECLARE 01 LE_FEEDBACK_CODE,
              03 MSGSEV       REAL FIXED BINARY(15,0),
              03 MSGNO        REAL FIXED BINARY(15,0),
              03 FLAGS,
                 05 CASE      BIT(2),
                 05 SEVERITY  BIT(3),
                 05 CONTROL   BIT(3),
              03 FACID        CHAR(3),
              03 ISI          REAL FIXED BINARY(31,0);

    DECLARE MSG_STRING       CHAR(255) VARYING;
    DECLARE MSG_DEST         REAL FIXED BINARY(31,0);
    DECLARE SUBSTR BUILTIN;
    DECLARE SYSPRINT FILE PRINT;

    MSG_DEST = 2;

    /***************************************/
    /* Clear all the environment variables */
    /***************************************/
```

```
   FUNC_CODE = 3;
   CALL CEEENV(FUNC_CODE,
               NAME_LEN,
               NAME,
               VALUE_LEN,
               VALUE_PTR,
               LE_FEEDBACK_CODE);
   /****************************************/
   /* Set an environment variable          */
   /****************************************/
   FUNC_CODE = 2;
   NAME = 'ENVAR1';
   VALUE = 'DEFAULT';
   VALUE_PTR = ADDR(VALUE);
   NAME_LEN = 6;
   VALUE_LEN = 7;
   CALL CEEENV(FUNC_CODE,
               NAME_LEN,
               NAME,
               VALUE_LEN,
               VALUE_PTR,
               LE_FEEDBACK_CODE);
   /****************************************/
   /* Get the variable, to see if added    */
   /****************************************/
   FUNC_CODE = 1;
   NAME = 'ENVAR1';
   VALUE = '';
   NAME_LEN = 6;
   VALUE_LEN = 0;
   CALL CEEENV(FUNC_CODE,
               NAME_LEN,
               NAME,
               VALUE_LEN,
               VALUE_PTR,
               LE_FEEDBACK_CODE);

   IF VALUE_LEN ^= 0 THEN DO;
       MSG_STRING = SUBSTR(NAME,1,NAME_LEN) ||
           '=' || SUBSTR(VALUE2,1,VALUE_LEN);
       PUT SKIP LIST(MSG_STRING);
   END;
   ELSE DO;
       MSG_STRING = 'SET REQUEST UNSUCCESSFUL';
       PUT SKIP LIST(MSG_STRING);
   END;
 END PLIENV;
```

CEEFMDA—Get default date format

CEEFMDA returns to the calling routine the default date picture string for a specified country. CEEFMDA is affected only by the country code setting of the COUNTRY runtime option or CEE3CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax

▶▶—CEEFMDA—(—*country_code*—,—*date_pic_str*—,—*fc*—)————————————▶◀

country_code **(input)**
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 481. The *country_code* is not case-sensitive. Also, if no value is specified, the default country code (as set by either the COUNTRY runtime option or the CEE3CTY callable service) is used. If you specify a *country_code* that is not valid, the default date format is 'YYYY-MM-DD'.

date_pic_str **(output)**
A fixed-length 80-character string (VSTRING), returned to the calling routine. It contains the default date picture string for the country specified. The picture string is left-justified and padded on the right with blanks if necessary.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3CB	2	3467	The country code *country_code* was invalid for CEEFMDA. The default date picture string *date_pic_string* was returned.

Usage notes

- z/OS UNIX considerations—CEEFMDA applies to the enclave. Every enclave has a single current country setting that has a single date format. Every thread in every enclave has the same default.

For more information

- For a list of the default settings for a specified country, see Table 32 on page 481.
- See "COUNTRY" on page 22 for an explanation of the COUNTRY runtime option.
- See "CEE3CTY—Set default country" on page 133 for an explanation of the CEE3CTY callable service.

Examples

1. Following is an example of CEEFMDA called by C/C++.

   C/C++ example of CEEFMDA

```
/*Module/File Name: EDCFMDA   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _CHAR2 country;
  _CHAR80 date_pic;

  /* get the default date format for Canada */
  memcpy(country,"CA",2);
  CEEFMDA(country,date_pic,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEEFMDA failed with message number %d\n",
           fc.tok_msgno);
    exit(2999);
  }
  /* print out the default date format for Canada */
  printf("%.80s\n",date_pic);
}
```

2. Following is an example of CEEFMDA called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTFMDA
      *******************************************
      **                                       **
      ** CBLFMDA - Call CEEFMDA to obtain the  **
      **           default date format         **
      *******************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLFMDA.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COUNTRY                PIC X(2).
       01  PICSTR                 PIC X(80).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info           PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLFMDA.
      ** Specify country code for the US and call
      **     CEEFMDA to return the default date format
      **     for the US.
           MOVE "US" TO COUNTRY.
           CALL "CEEFMDA" USING COUNTRY , PICSTR , FC.

      ** If CEEFMDA runs successfully, display result
           IF CEE000 of FC THEN
               DISPLAY "The default date format for "
                   "the US is: " PICSTR
           ELSE
               DISPLAY "CEEFMDA failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
           GOBACK.
```

3. Following is an example of CEEFMDA called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMFMDA                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEEFMDA - obtain default date        **/
 /**                    format                      **/
 /**                                                **/
 /****************************************************/

 PLIFMDA: PROC OPTIONS(MAIN);
```

```
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
    DCL COUNTRY CHARACTER ( 2 );
    DCL PICSTR CHAR(80);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    COUNTRY = 'US';   /* Specify country code for    */
                      /*    the United States        */

    /* Get the default date format for the US        */
    CALL CEEFMDA ( COUNTRY , PICSTR , FC );

    /* Print the default date format for the US      */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'The default date format for the US is '
          || PICSTR );
       END;
    ELSE  DO;
       DISPLAY( 'CEEFMDA failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLIFMDA;
```

CEEFMDT—Get default date and time format

CEEFMDT returns the default date and time picture strings for the country specified by *country_code*. CEEFMDT is affected only by the country code setting of the COUNTRY runtime option or CEE3CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax

```
►►──CEEFMDT──(──country_code──,──datetime_str──,──fc──)────────────────────►◄
```

country_code **(input)**
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 481. The *country_code* is not case-sensitive. Also, if no value is specified, the default country code (as set by either the COUNTRY runtime option or by CEE3CTY) is used. If you specify a *country_code* that is not valid, the default date/time picture string is `'YYYY-MM-DD HH:MI:SS'`.

datetime_str **(output)**
A fixed-length 80-character string (VSTRING), returned to the calling routine. It contains the default date and time picture string for the country specified. The picture string is left-justified and padded on the right with blanks, if necessary.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3CF	2	3471	The country code *country_code* was invalid for CEEFMDT. The default date and time picture string *datetime_str* was returned.

Usage notes

- z/OS UNIX considerations—CEEFMDT applies to the enclave. Every enclave has a single current country setting that has a single date and time format. Every thread in every enclave has the same default.

For more information

- For a list of the default settings for a specified country, see Table 32 on page 481.
- See "COUNTRY" on page 22 for an explanation of the COUNTRY runtime option.
- See "CEE3CTY—Set default country" on page 133 for an explanation of CEE3CTY.

Examples

1. Following is an example of CEEFMDT called by C/C++.

```
/*Module/File Name: EDCFMDT   */

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country;
   _CHAR80 date_pic;

   /* get the default date and time format for Canada */
   memcpy(country,"CA",2);
   CEEFMDT(country,date_pic,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEFMDT failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   /* print out the default date and time format */
   printf("%.80s\n",date_pic);
}
```

2. Following is an example of CEEFMDT called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTFMDT
      *************************************************
      **                                             **
      ** CBLFMDT - Call CEEFMDT to obtain the        **
      **          default date & time format        **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLFMDT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COUNTRY                PIC X(2).
       01  PICSTR                 PIC X(80).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info           PIC S9(9) BINARY.

       PROCEDURE DIVISION.
       PARA-CBLFMDT.
      ** Specify country code for the US
           MOVE "US" TO COUNTRY.
      ** Call CEEFMDT to return the default date and
      **     time format for the US
           CALL "CEEFMDT" USING COUNTRY , PICSTR , FC.

      ** If CEEFMDT runs successfully, display result.
           IF CEE000 of FC THEN
               DISPLAY "The default date and time "
                   "format for the US is: " PICSTR
             ELSE
               DISPLAY "CEEFMDT failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
           GOBACK.
```

3. Following is an example of CEEFMDT called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMFMDT                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEEFMDT - obtain default             **/
 /**                     date & time format         **/
 /**                                                **/
 /****************************************************/

 PLIFMDT: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL COUNTRY CHARACTER ( 2 );
    DCL PICSTR  CHAR(80);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
```

```
         03 ISI   /* Instance-Specific Information */
                     REAL FIXED BINARY(31,0);

 COUNTRY = 'US';  /* Specify country code for     */
                  /* the United States            */

 /* Call CEEFMDT to get default date format       */
 /*    for the US                                 */
 CALL CEEFMDT ( COUNTRY , PICSTR , FC );

 /* Print default date format for the US          */
 IF  FBCHECK( FC, CEE000)  THEN  DO;
    PUT SKIP LIST( 'The default date and time '
       || 'format for the US is ' || PICSTR );
    END;
 ELSE  DO;
    DISPLAY( 'CEEFMDT failed with msg '
       || FC.MsgNo );
    STOP;
    END;


END PLIFMDT;
```

CEEFMON—Format monetary string

CEEFMON, analogous to the C function `strfmon()`, converts numeric values to monetary strings according to the specifications passed to it. These specifications work in conjunction with the format conversions set in the current locale. The current locale's LC_MONETARY category affects the behavior of this service, including the monetary radix character, the thousands separator, the monetary grouping, the currency symbols (national and international), and formats.

CEEFMON is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►──CEEFMON──(──omitted_parm──,──stringin──,──maxsize──,──format──,───────────►

►──stringout──,──result──,──fc──)────────────────────────────────────────────►◄
```

omitted_parm
: This parameter is reserved for future expansion and must be omitted. For information about how to code an omitted parm, see “Invoking callable services” on page 114.

stringin **(input)**
: An 8-byte field that contains the value of a double-precision floating point number.

maxsize **(input)**
: A 4-byte integer that specifies the maximum number of bytes (including the string terminator) that can be placed in *stringout*.

format **(input)**
: A halfword length-prefixed character string (VSTRING) of 256 bytes that contains plain characters and a conversion specification. Plain characters are

copied to the output stream. Conversion specification results in the fetching of the input *stringin* argument that is converted and formatted.

A conversion specification consists of a percent character (%), optional flags, optional field width, optional left precision, optional right precision, and a required conversion character that determines the conversion to be performed.

Flags (optional)
You can specify one or more of the following flags to control the conversion.

=*f* An = followed by a single character *f* specifies that the character *f* should be used as the numeric fill character. The default numeric fill character is the space character. This option does not affect the other fill operations (such as field-width filling), which always use the space as the fill character.

^ Do not format the currency amount with the grouping characters. The default is to insert the grouping characters if defined for the current locale.

+ or (Specifies the style of representing positive and negative currency amounts. You must specify either + or *(*. If + is specified, the locale's equivalent of + and - are used (for example, in USA: the empty (null) string if positive and - (minus sign) if negative). If *(* is specified, the locale's equivalent of enclosing negative amounts within a parenthesis is used. If this option is not included, a default specified by the current locale is used.

! Suppresses the currency symbol from the output conversion.

Field Width (optional)
A decimal digit string *w* that specifies a minimum field width in which the result of the conversion is right-justified (or left-justified if *-w* is specified).

Left Precision (optional)
A # (pound sign) followed by the decimal digit string *n*, (specified as #*n*), indicates a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to the CEEFMON service aligned in the same columns. It can also be used to fill unused positions with a special character as in $***123.45. This option causes an amount to be formatted as if it has the number of digits specified by *n*. If more digit positions are required than are specified, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character. See the *=f* specification above.

If grouping is enabled, it is applied to the combined fill characters plus regular digits. However, if the fill character is not 0 (digit zero), grouping separators inserted after a fill character are replaced by the same fill character ($0,001,234.56 versus $****1,234.56).

Right Precision (optional)
A period (.) followed by the decimal digit string *p*, (specified as *.p*), indicates the number of digits after the radix character. If the value of the precision *p* is zero, no radix character appears. If this option is not included, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting.

Conversion Characters (required)
One of the following conversion characters must be specified.

i	The double argument is formatted according to the locale's international currency format (for example, in USA: USD 1,234.56).
n	The double argument is formatted according to the locale's national currency format (for example, in USA: $1,234.56).
%	No argument is converted; the conversion specification %% is replaced by a single %.

stringout **(output)**
Contains the output of the CEEFMON service.

result **(output)**
Contains the number of characters placed in *stringout*, if successful. If unsuccessful, an error is reported.

fc **(output/optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.
CEE3VM	3	4086	Input Error: The number of characters to be formatted must be greater than zero.

Usage notes

- PL/I MTF consideration—CEEFMON is not supported in PL/I MTF applications.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- For more information about the `setlocale()` function, see "COUNTRY" on page 22, "CEE3CTY—Set default country" on page 133, and "CEE3LNG—Set national language" on page 172.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 411.

Examples

1. An example of CEEFMON called by COBOL:

```
 CBL LIB,QUOTE
*Module/File Name: IGZTFMON
*************************************************
*  Example for callable service CEEFMON        *
*   Function: Convert a numeric value to a     *
*             monetary string using specified  *
*             format passed as parameter.      *
*   Valid only for COBOL for MVS & VM Release 2 *
*   or later.                                  *
*************************************************
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBFMON.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  Monetary       COMP-2.
 01  Max-Size       PIC S9(9)  BINARY.
 01  Format-Mon.
     02  FM-Length  PIC S9(4)  BINARY.
     02  FM-String  PIC X(256).
 01  Output-Mon.
     02  OM-Length  PIC S9(4)  BINARY.
     02  OM-String  PIC X(60).
 01  Length-Mon     PIC S9(9)  BINARY.
 01  Locale-Name.
     02  LN-Length  PIC S9(4)  BINARY.
     02  LN-String  PIC X(256).
**   Use Locale category constants
 COPY CEEIGZLC.
 01  FC.
     02  Condition-Token-Value.
     COPY CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
**

 PROCEDURE DIVISION.
** Set up locale name for United States
     MOVE 14 TO LN-Length.
     MOVE "En_US.IBM-1047"
           TO LN-String (1:LN-Length).
** Set all locale categories to United States.
** Use LC-ALL category constant from CEEIGZLC.
     CALL "CEESETL" USING Locale-Name, LC-ALL, FC.
** Check feedback code
     IF Severity > 0
        DISPLAY "Call to CEESETL failed. " Msg-No
        STOP RUN
     END-IF.  ** Set up numeric value
     MOVE 12345.62 TO Monetary.
     MOVE 60 TO Max-Size.
     MOVE 2 TO FM-Length.
     MOVE "%i" TO FM-String (1:FM-Length).
** Call CEEFMON to convert numeric value
     CALL "CEEFMON" USING OMITTED, Monetary,
                          Max-Size, Format-Mon
                          Output-Mon, Length-Mon,
                          FC.

** Check feedback code and display result
     IF Severity > 0
        DISPLAY "Call to CEEFMON failed. " Msg-No
     ELSE
        DISPLAY "International format is "
                OM-String(1:OM-Length)
     END-IF.

     STOP RUN.
 END PROGRAM COBFMON.
```

2. An example of CEEFMON called by PL/I:

```
*PROCESS MACRO;
 /*Module/File Name: IBMFMON                           */
 /****************************************************/
 /* Example for callable service CEEFMON             */
 /* Function: Convert a numeric value to a monetary  */
 /*  string using specified format passed as parm    */
 /****************************************************/

 PLIFMON: PROC OPTIONS(MAIN);

 %INCLUDE CEEIBMAW; /* ENTRY defs, macro defs        */
 %INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants   */
 %INCLUDE CEEIBMLC; /* Locale category constants     */

 /* CEESETL service call arguments */
 DCL LOCALE_NAME CHAR(14) VARYING;

 /* CEEFMON service call arguments */
 DCL MONETARY REAL FLOAT DEC(16);   /* input value   */
 DCL MAXSIZE_FMON BIN FIXED(31);    /* output size   */
 DCL FORMAT_FMON CHAR(256) VARYING; /* format spec   */
 DCL RESULT_FMON BIN FIXED(31);     /* result status */
 DCL OUTPUT_FMON CHAR(60) VARYING;  /* output string */

 DCL 01 FC,                        /* Feedback token */
        03 MsgSev    REAL FIXED BINARY(15,0),
        03 MsgNo     REAL FIXED BINARY(15,0),
        03 Flags,
           05 Case      BIT(2),
           05 Severity  BIT(3),
           05 Control   BIT(3),
        03 FacID     CHAR(3),       /* Facility ID */
        03 ISI      /* Instance-Specific Information */
                     REAL FIXED BINARY(31,0);

   /* init locale name to United States */
   LOCALE_NAME = 'En_US.IBM-1047';

   /* use LC_ALL category constant from CEEIBMLC */
   CALL CEESETL (LOCALE_NAME, LC_ALL, FC);

   /* FBCHECK macro used (defined in CEEIBMCT) */
   IF FBCHECK (FC, CEE2KE) THEN
     DO;   /* invalid locale name */
       DISPLAY ('Locale LC_ALL Call '||FC.MsgNo);
       STOP;
     END;MONETARY = 12345.62; /* monetary numeric value    */
   MAXSIZE_FMON = 60;   /* max char length returned  */
   FORMAT_FMON = '%i';  /* international currency    */

   CALL CEEFMON ( *, /* optional argument           */
     MONETARY ,      /* input, 8 byte floating point */
     MAXSIZE_FMON,   /* maximum size of output string*/
     FORMAT_FMON,    /* conversion request           */
     OUTPUT_FMON,    /* string returned by CEEFMON   */
     RESULT_FMON,    /* no. of chars in OUTPUT_FMON  */
     FC );           /* feedback code structure      */

   IF RESULT_FMON = -1 THEN
     DO;
       /* FBCHECK macro used (defined in CEEIBMCT) */
       IF FBCHECK( FC, CEE3VM ) THEN
         DISPLAY ( 'Invalid input '||MONETARY );
       ELSE
         DISPLAY ('CEEFMON not completed '||FC.MsgNo );
       STOP;
     END;
```

```
         ELSE
           DO;
             PUT SKIP LIST(
               'International Format '||OUTPUT_FMON );
           END;

       END PLIFMON;
```

CEEFMTM—Get default time format

CEEFMTM returns to the calling routine the default time picture string for the country specified by *country_code*. For a list of the default settings for a specified country, see Table 32 on page 481. CEEFMTM is affected only by the country code setting of the COUNTRY runtime option or CEE3CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax

```
►►—CEEFMTM—(—country_code—,—time_pic_str—,—fc—)——————————————————►◄
```

country_code **(input)**
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 481. The *country_code* is not case-sensitive. Also, if no value is specified, the default country code (as set by either the COUNTRY runtime option or the CEE3CTY callable service) is used. If you specify a *country_code* that is not valid, the default time picture string is `'HH:MI:SS'`.

time_pic_str **(output)**
A fixed-length 80-character string (VSTRING), representing the default time picture string for the country specified. The picture string is left-justified and padded on the right with blanks if necessary.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3CD	2	3469	The country code *country_code* was invalid for CEEFMTM. The default time picture string *time_pic_string* was returned.
CEE3CE	2	3470	The date and time string *datetime_str* was truncated and was not defined in CEEFMDT.

Usage notes

- z/OS UNIX considerations—CEEFMTM applies to the enclave. Every enclave has a single current country setting that has a single time format. Every thread in every enclave has the same default.

For more information

- See "COUNTRY" on page 22 for an explanation of the COUNTRY runtime option.
- See "CEE3CTY—Set default country" on page 133 for an explanation of the CEE3CTY callable service.

Examples

1. Following is an example of CEEFMTM called by C/C++.

```
/*Module/File Name: EDCFMTM   */

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _CHAR2 country;
   _CHAR80 time_pic;

   /* get the default time format for Canada */
   memcpy(country,"CA",2);
   CEEFMTM(country,time_pic,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEFMTM failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   /* print out the default time format */
   printf("%.80s\n",time_pic);
}
```

2. Following is an example of CEEFMTM called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTFMTM
      *************************************************
      **                                             **
      ** IGZTFMTM - Call CEEFMTM to obtain the       **
      **           default time format               **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTFMTM.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COUNTRY                PIC X(2).
       01  PICSTR                 PIC X(80).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info           PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLFMTM.
      ** Specify country code for the US.
           MOVE "US" TO COUNTRY.

      ** Call CEEFMTM to return the default time format
      ** for the US.
           CALL "CEEFMTM" USING COUNTRY , PICSTR , FC.

      ** If CEEFMTM runs successfully, display result.
           IF CEE000 of FC  THEN
               DISPLAY  "The default time format for "
                   "the US is: " PICSTR
           ELSE
               DISPLAY "CEEFMTM failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
           GOBACK.
```

3. Following is an example of CEEFMTM called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMFMTM                         */
 /****************************************************/
 /**                                                **/
 /** Function: CEEFMTM - obtain default time        **/
 /**                     format                     **/
 /**                                                **/
 /** This example calls CEEFMTM to request the      **/
 /** default time format for the US and print       **/
 /** it out.                                        **/
 /**                                                **/
 /****************************************************/
 PLIFMTM: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL COUNTRY CHARACTER ( 2 );
    DCL PICSTR CHAR(80);
    DCL 01 FC,                      /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    COUNTRY = 'US'; /* Specify country code for      */
                    /*    the United States          */

    /* Call CEEFMTM to get default time format for   */
    /*    the US                                     */
    CALL CEEFMTM ( COUNTRY , PICSTR , FC );

    /* Print the default time format for the US      */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'The default time format for the US is '
          || PICSTR);
       END;
    ELSE  DO;
       DISPLAY( 'CEEFMTM failed with msg '
          || FC.MsgNo );
```

```
                    STOP;
                    END;

          END PLIFMTM;
```

CEEFRST—Free heap storage

CEEFRST frees storage previously allocated by CEEGTST or by a language intrinsic function. Normally, you do not need to call CEEFRST because Language Environment automatically returns all heap storage to the operating system when the enclave terminates. However, if you are allocating a large amount of heap storage, you should free the storage when it is no longer needed. This freed storage then becomes available for later requests for heap storage, thus reducing the total amount of storage needed to run the application.

All requests for storage are conditional. If storage is not available, the feedback code (*fc*) is set and returned to you, but the thread does **not** abend. An attempt to free storage that was already marked as free produces no action and returns a non-CEE000 symbolic feedback code. An attempt to free storage at anything other than a valid starting address produces no action and returns a non-CEE000 symbolic feedback code. The application does not abend.

However, if you call CEEFRST for an invalid address, and you had specified TRAP(OFF), your application **can** abend. Language Environment's reaction to this is undefined. Also, partial freeing of an allocated area is not supported.

When storage is allocated by CEEGTST, its allocated size is used during free operations. Storage allocated by CEEGTST, but not explicitly freed, is automatically freed at enclave termination.

CEEFRST generates a system-level free storage call to return a storage increment to the operating system only when:

- The last heap element within an increment is being freed, and
- The HEAP runtime option or a call to CEECRHP specifies FREE (note that KEEP is the IBM-supplied default setting for the initial heap).

Otherwise, the freed storage is simply added to the free list; it is not returned to the operating system until termination. The out-of-storage condition can cause freeing of empty increments even when KEEP is specified.

Syntax

```
►►──CEEFRST──(──address──,──fc──)──────────────────────────────────────►◄
```

address **(input)**
: A fullword address pointer. *address* is the address returned by a previous CEEGTST call or a language intrinsic function such as ALLOCATE or `malloc()`. The storage at this address is deallocated.

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result

of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following feedback codes can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0PA	3	0810	The storage address in a free storage (CEEFRST) request was not recognized, or heap storage (CEECZST) control information was damaged.

Usage notes

- If you specify *heap_free_value* in the STORAGE runtime option, all freed storage is overwritten with *heap_free_value*. Otherwise, it is simply marked as available.

  Portions of the freed storage area can be used to hold internal storage manager control information. These areas are overwritten, but not with *heap_free_value*.
- The heap identifier is inferred from the address of the storage to be freed. The storage is freed from the heap in which it was allocated.
- The address passed as the argument is a dangling pointer after a call to CEEFRST. The storage freed by this operation can be reallocated on a subsequent CEEGTST call. If the pointer is not reassigned, any further use of it causes unpredictable results.
- z/OS UNIX considerations—CEEFRST applies to the enclave. One thread can allocate storage, and another can free it.

For more information

- See “CEEGTST—Get heap storage” on page 313 for more information about the CEEGTST callable service.
- See “HEAP” on page 36 for further information about the HEAP runtime option.
- See “CEECRHP—Create new additional heap” on page 221 for more information about the CEECRHP callable service.
- See “STORAGE” on page 79 for further information about the STORAGE runtime option.

Examples

1. An example of CEEFRST called by C/C++:

```
/*Module/File Name: EDCFRST   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _INT4 heapid, size;
   _POINTER address;
   _FEEDBACK fc;
 /*  .
     .
     .  */
   heapid = 0;    /* get storage from initial heap   */
```

```
  size = 4000;   /* number of bytes of heap storage */

  /* obtain the storage using CEEGTST */
  CEEGTST(&heapid,&size,&address,&fc);

  /* check the first 4 bytes of the feedback token */
  /* (0 if successful) */
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEGTST failed with message number %d\n",
            fc.tok_msgno);
     exit(99);
  }
/*  .
    .
    .  */

  /* free the storage that was previously obtained */
  /* using CEEGTST */
  CEEFRST(&address,&fc);

  /* check the first 4 bytes of the feedback token */
  /* (0 if successful) */
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEFRST failed with message number %d\n",
            fc.tok_msgno);
     exit(99);
  }
/*  .
    .
    .  */
}
```

2. An example of CEEFRST called by COBOL:

```
CBL LIB,QUOTE
      *Module/File Name: IGZTFRST
      *************************************************
      **                                             **
      ** IGZTFRST - Call CEEFRST to free heap        **
      **            storage                          **
      **                                             **
      ** In this example, a call is made to          **
      ** CEEGTST to obtain 4000 bytes of storage     **
      ** from the initial heap (HEAPID = 0).         **
      ** A call is then made to CEEFRST to free      **
      ** the storage.                                **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTFRST.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  HEAPID                  PIC S9(9) BINARY.
       01  STGSIZE                 PIC S9(9) BINARY.
       01  ADDRSS                  USAGE IS POINTER.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
```

```
 PROCEDURE DIVISION.
 PARA-CBLFRST.

** Specify 0 to get storage from the initial
**    heap.
** Specify 4000 to get 4000 bytes of storage.
** Call CEEGTST to obtain storage.
     MOVE 0 TO HEAPID.
     MOVE 4000 TO STGSIZE.

     CALL "CEEGTST" USING HEAPID , STGSIZE ,
         ADDRSS , FC.
     IF CEE000 of FC  THEN
         DISPLAY "Obtained " STGSIZE " bytes of"
             " storage at location " ADDRSS
             " from heap number " HEAPID
     ELSE
         DISPLAY "CEEGTST failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
** To free storage, use the address returned
**     by CEECRHP in the call to CEEFRST.
     CALL "CEEFRST" USING ADDRSS , FC.
     IF CEE000 of FC THEN
         DISPLAY "Returned " STGSIZE " bytes of"
             " storage at location " ADDRSS
             "  to  heap number " HEAPID
     ELSE
         DISPLAY "CEEFRST failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.
     GOBACK.
```

3. An example of CEEFRST called by PL/I:

```
*PROCESS MACRO;
 /*Module/File Name: IBMFRST                       */
 /*************************************************/
 /**                                             **/
 /** Function: CEEFRST - free heap storage       **/
 /**                                             **/
 /** This example calls CEEGTST to obtain storage **/
 /** from the initial heap, and then calls       **/
 /** CEEFRST to discard it.                      **/
 /**                                             **/
 /*************************************************/
 PLIFRST: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL ADDRSS   POINTER;
    DCL HEAPID   REAL FIXED BINARY(31,0);
    DCL STGSIZE  REAL FIXED BINARY(31,0);
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),    /* Facility ID */
           03 ISI    /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    DCL 01 FC2,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
         03 MsgNo     REAL FIXED BINARY(15,0),
         03 Flags,
            05 Case      BIT(2),
            05 Severity  BIT(3),
            05 Control   BIT(3),
         03 FacID     CHAR(3),    /* Facility ID */
         03 ISI   /* Instance-Specific Information */
                      REAL FIXED BINARY(31,0);

  HEAPID = 0; /* get storage from the initial heap */

  STGSIZE = 4000;   /* get 4000 bytes of storage   */

  /* Call CEEGTST to obtain the storage            */
  CALL CEEGTST ( HEAPID, STGSIZE, ADDRSS, FC );
  IF  FBCHECK( FC, CEE000)  THEN  DO;
     PUT SKIP LIST( 'Obtained ' || STGSIZE
        || ' bytes of storage at location '
        || DECIMAL( UNSPEC( ADDRSS ) )
        || ' from heap ' || HEAPID );
     END;

  ELSE  DO;
     DISPLAY( 'CEEGTST failed with msg '
        || FC.MsgNo );
     STOP;
     END;  /* Call CEEFRST with the address returned from */
  /*    CEEGTST to free the storage allocated by   */
  /*    the call to CEEGTST                        */
  CALL CEEFRST ( ADDRSS, FC2 );
  IF  FBCHECK( FC2, CEE000)  THEN  DO;
     PUT SKIP LIST( 'Storage block at location '
        || DECIMAL( UNSPEC( ADDRSS ) ) || ' freed');
     END;
  ELSE  DO;
     DISPLAY( 'CEEFRST failed with msg '
        || FC2.MsgNo );
     STOP;
     END;

END PLIFRST;
```

CEEFTDS—Format time and date into character string

CEEFTDS, analogous to the C function `strftime()`, converts the internal time and date to character strings according to the specifications passed to it by the TD_STRUCT. These specifications work in conjunction with the format conversions set in the current locale. The current locale's LC_TIME category affects the behavior of this service, including the actual values for the format specifiers. CEEFTDS is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►──CEEFTDS──(──omitted_parm──,──time_and_date──,──maxsize──,──format──,──────────►

►──stringout──,──fc──)────────────────────────────────────────────────────────────►◄
```

omitted_parm
: This parameter is reserved for future expansion and must be omitted. For information about how to code an omitted parm, see "Invoking callable services" on page 114.

time_and_date **(input)**
: Points to the time structure that is to be converted. The structure has the following format:

  td_sec
  : A 4-byte integer that describes the number of seconds in the range of 0 through 59.

  td_min
  : A 4-byte integer that describes the number of minutes in the range of 0 through 59.

  td_hour
  : A 4-byte integer that describes the number of hours in the range of 0 through 23.

  td_mday
  : A 4-byte integer that describes the day of the month in the range of 1 through 31.

  td_mon
  : A 4-byte integer that describes the month of the year in the range of 0 through 12.

  td_year
  : A 4-byte integer that describes the year of an era.

  td_wday
  : A 4-byte integer that describes the day of the week in the range of 0 through 6.

  td_yday
  : A 4-byte integer that describes the day of the year in the range of 0 through 365.

  td_isdst
  : A 4-byte integer that is a flag for daylight savings time.

maxsize **(input)**
: A 4-byte integer that specifies the maximum length (including the string terminator) of the string that can be placed in *stringout*.

format **(input)**
: A halfword length-prefixed character string (VSTRING) of 256 bytes that contains the conversion specifications. The format parameter is a character string containing two types of objects: plain characters that are placed in the output string and conversion specifications that convert information from *time_and_date* into readable form in the output string. Each conversion specification is a sequence of this form:

  `%[-][`*width*`][.`*precision*`]`*type*

  %
  : A percent sign (%) introduces a conversion specification. This character is required.

  width **(optional)**
  : An optional decimal digit string that specifies a minimum field width. A converted value that has fewer characters than the field width is

padded with spaces to the left. If the decimal digit string is preceded by a minus sign, padding with spaces occurs to the right of the converted value.

If no width is given, for numeric fields the appropriate default width is used with the field padded on the left with zeros, as required. For strings, the output field is made exactly wide enough to contain the string.

precision **(optional)**
An optional value that specifies the maximum number of characters to be printed for the conversion specification. The precision value is a decimal digit string preceded by a period. If the value to be output is longer than the precision, it is truncated on the right.

type Specifies the type of conversion to be performed. The characters and their meanings are:

%a The abbreviated weekday name (for example, Sun) defined by the *abday* statement in the current locale.

%A The full weekday name (for example, Sunday) defined by the *day* statement in the current locale.

%b The abbreviated month name (for example, Jan) defined by the *abmon* statement in the current locale.

%B The full month name (for example, January) defined by the *month* statement in the current locale.

%c The date and time format defined by the *d_t_fmt* statement in the current locale.

%d The day of the month as a decimal number (01 to 31).

%D The date in *%m/%d/%y* format (for example, 01/31/91).

%e The day of the month as a decimal number (1 to 31). The *%e* field descriptor uses a two-digit field. If the day of the month is not a two-digit number, the leading digit is filled with a space character.

%E The combined alternative era year and name, respectively, in *%o %N* format in the current locale.

%F The ISO 8601:2000 standard date format, equivalent to *%Y-%m-%d*.

%g The last 2 digits of the week-based year as a decimal number [00,99].

%G The week-based year as a 4-digit decimal number.

%h The abbreviated month name (for example, Jan) defined by the *abmon* statement in the current locale. This field descriptor is a synonym for the *%b* field descriptor.

%H The 24-hour clock hour as a decimal number (00 to 23).

%I The 12-hour clock hour as a decimal number (01 to 12).

%j The day of the year as a decimal number (001 to 366).

%m The month of the year as a decimal number (01 to 12).

%M The minutes of the hour as a decimal number (00 to 59).

%n	Specifies a new-line character.
%N	The alternative era name in the current locale.
%o	The alternative era year in the current locale.
%p	The AM or PM string defined by the *am_pm* statement in the current locale.
%r	The 12-hour clock time with AM/PM notation as defined by the *t_fmt_ampm* statement *(%I:%M:%S [AM|PM])* in the current locale.
%S	The seconds of the minute as a decimal number (00 to 60).
%t	Specifies a tab character.
%T	Represents 24-hour clock time in the format *%H:%M:%S* (for example, 16:55:15).
%U	The week of the year as a decimal number (00 to 53). Sunday, or its equivalent as defined by the *day* statement, is considered the first day of the week for calculating the value of this field descriptor.
%w	The day of the week as a decimal number (0 to 6). Sunday, or its equivalent as defined by the *.day* statement, is considered as 0 for calculating the value of this field descriptor.
%W	The week of the year as a decimal number (00 to 53). Monday, or its equivalent as defined by the *day* statement, is considered the first day of the week for calculating the value of this field descriptor.
%x	The date format defined by the *d_fmt* statement in the current locale.
%X	The time format defined by the *t_fmt* statement in the current locale.
%y	The year of the century (00 to 99).
%Y	The year as a decimal number (for example, 1989).
%z	The offset from UTC in ISO8601:2000 standard format (*+hhmm* or *-hhmm*). For example, "-0430" means 4 hours 30 minutes behind UTC (west of Greenwich).
%Z	The time-zone name, if one can be determined (for example, EST); no characters are displayed if a time zone cannot be determined.
%%	Specifies a percent sign (%) character.

stringout **(output)**
A halfword length-prefixed character string (VSTRING) of 256 bytes that contains the formatted time and date output of the CEEFTDS service.

fc **(output/optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.
CEE3VM	3	4086	Input Error: The number of characters to be formatted must be greater than zero.

Usage notes

- PL/I MTF consideration—CEEFTDS is not supported in PL/I MTF applications.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- For more information about the `setlocale()` function, see “COUNTRY” on page 22, “CEE3CTY—Set default country” on page 133, and “CEE3LNG—Set national language” on page 172.
- For more information about the CEESETL callable service, see “CEESETL—Set locale operating environment” on page 411.

Examples

1. An example of CEEFTDS called by COBOL:

```
 CBL LIB,QUOTE
*Module/File Name: IGZTFTDS
*************************************************
*  Example for callable service CEEFTDS        *
*  Function: Convert numeric time and date     *
*            values to a string using specified *
*            format string and locale format    *
*            conversions.                       *
*  Valid only for COBOL for MVS & VM Release 2  *
*  or later.                                    *
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  MAINFTDS.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
* Use TD-Struct for CEEFTDS calls
 COPY CEEIGZTD.
*

 PROCEDURE DIVISION.
*  Subroutine needed for pointer addressing
     CALL "COBFTDS" USING TD-Struct.

     STOP RUN.
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBFTDS.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
* Use Locale category constants
 COPY CEEIGZLC.
*
 01  Ptr-FTDS  POINTER.
 01  Output-FTDS.
     02  O-Length  PIC S9(4)  BINARY.
     02  O-String  PIC X(72).
 01  Format-FTDS.
     02  F-Length  PIC S9(4)  BINARY.
     02  F-String  PIC X(64).
 01  Max-Size  PIC S9(9)  BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity  PIC S9(4) BINARY.
             04  Msg-No    PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl  PIC X.
         03  Facility-ID   PIC XXX.
     02  I-S-Info  PIC S9(9) BINARY.
 LINKAGE SECTION.
* Use TD-Struct for calls to CEEFTDS
 COPY CEEIGZTD.  *
 PROCEDURE DIVISION USING TD-Struct.
*  Set up time and date values
     MOVE 1 TO TM-Sec.
     MOVE 2 TO TM-Min.
     MOVE 3 TO TM-Hour.
     MOVE 9 TO TM-Day.
     MOVE 11 TO TM-Mon.
     MOVE 94 TO TM-Year.
     MOVE 5 TO TM-Wday.
     MOVE 344 TO TM-Yday.
     MOVE 1 TO TM-Is-DLST.

*  Set up format string for CEEFTDS call
     MOVE 72 TO Max-Size.
     MOVE 36 TO F-Length.
     MOVE "Today is %A, %b %d  Time:  %I:%M  %p"
        TO F-String (1:F-Length).

*  Set up pointer to structure for CEEFTDS call
     SET Ptr-FTDS TO ADDRESS OF TD-Struct.

*  Call CEEFTDS to convert numeric values
     CALL "CEEFTDS" USING OMITTED, Ptr-FTDS,
                          Max-Size, Format-FTDS,
                          Output-FTDS, FC.

*  Check feedback code and display result
     IF Severity = 0
        DISPLAY "Format " F-String (1:F-Length)
        DISPLAY "Result " O-String (1:O-Length)
     ELSE
        DISPLAY "Call to CEEFTDS failed. " Msg-No
     END-IF.

     EXIT PROGRAM.
 END PROGRAM COBFTDS.
*
 END PROGRAM MAINFTDS.
```

2. An example of CEEFTDS called by PL/I:

```
*PROCESS MACRO;
 /*Module/File Name: IBMFTDS                          */
 /****************************************************/
 /* Example for callable service CEEFTDS             */
 /* Function: Convert numeric time and date values   */
 /*  to a string based on a format specification     */
 /*  string parameter and locale format conversions  */
 /****************************************************/

 PLIFTDS: PROC OPTIONS(MAIN);
 %INCLUDE CEEIBMAW; /* ENTRY defs, macro defs        */
 %INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants   */
 %INCLUDE CEEIBMLC; /* Locale category constants     */
 %INCLUDE CEEIBMTD; /* TD_STRUCT for CEEFTDS calls   */

 /* use explicit pointer to local TD_STRUCT structure*/
 DCL TIME_AND_DATE POINTER INIT(ADDR(TD_STRUCT));

 /* CEEFTDS service call arguments */
 DCL MAXSIZE_FTDS BIN FIXED(31); /* OUTPUT_FTDS size */
 DCL FORMAT_FTDS CHAR(64) VARYING;  /* format string */
 DCL OUTPUT_FTDS CHAR(72) VARYING;  /* output string */DCL 01 FC,
/* Feedback token */
        03 MsgSev    REAL FIXED BINARY(15,0),
        03 MsgNo     REAL FIXED BINARY(15,0),
        03 Flags,
           05 Case      BIT(2),
           05 Severity  BIT(3),
           05 Control   BIT(3),
        03 FacID     CHAR(3),       /* Facility ID */
        03 ISI      /* Instance-Specific Information */
                     REAL FIXED BINARY(31,0);

   /* specify numeric input fields for conversion    */
   TD_STRUCT.TM_SEC=1;   /* seconds after min (0-61) */
   TD_STRUCT.TM_MIN=2;   /* minutes after hour (0-59)*/
   TD_STRUCT.TM_HOUR=3; /* hours since midnight(0-23)*/
   TD_STRUCT.TM_MDAY=9;  /* day of the month (1-31)  */
   TD_STRUCT.TM_MON=11;  /* months since Jan(0-11)   */
   TD_STRUCT.TM_YEAR=94; /* years since 1900         */
   TD_STRUCT.TM_WDAY=5;  /* days since Sunday (0-6)  */
   TD_STRUCT.TM_YDAY=344;/* days since Jan 1 (0-365) */
   TD_STRUCT.TM_ISDST=1; /* Daylight Saving Time flag*/

   /* specify format string for CEEFTDS call         */
   FORMAT_FTDS = 'Today is %A, %b %d  Time: %I:%M %p';

   MAXSIZE_FTDS = 72; /* specify output string size  */

   CALL CEEFTDS ( *, TIME_AND_DATE, MAXSIZE_FTDS,
                  FORMAT_FTDS, OUTPUT_FTDS, FC );

   /* FBCHECK macro used (defined in CEEIBMCT) */
   IF FBCHECK( FC, CEE000 ) THEN
     DO; /* CEEFTDS call is successful */
       PUT SKIP LIST('Format '||FORMAT_FTDS );
       PUT SKIP LIST('Results in '||OUTPUT_FTDS );
     END;
   ELSE
     DISPLAY ( 'Format '||FORMAT_FTDS||
               ' Results in '||FC.MsgNo );

 END PLIFTDS;
```

CEEGMT—Get current Greenwich Mean Time

CEEGMT returns the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. The returned values are compatible with those generated and used by the other Language Environment date and time services. In order for the results of this service to be meaningful, your system's TOD (time-of-day) clock must be set to Greenwich Mean Time and be based on the standard epoch. Use CEEGMTO to get the offset from GMT to local time.

The values returned by CEEGMT are handy for elapsed time calculations. For example, you can calculate the time elapsed between two calls to CEEGMT by calculating the differences between the returned values.

Language Environment treats Coordinated Universal Time (UTC) and Greenwich Mean Time (GMT) as the same. You can use the CEEUTC service, which is an alias of the CEEGMT service, to get the same value.

Syntax

```
►►—CEEGMT—(—output_GMT_Lilian—,—output_GMT_seconds—,—fc—)————————————►◄
```

output_GMT_Lilian **(output)**
A 32-bit binary integer representing the current time in Greenwich, England, in the Lilian format (the number of days since 14 October 1582). For example, 16 May 1988 is day number 148138. If GMT is not available from the system, *output_GMT_Lilian* is set to 0 and CEEGMT terminates with a non-CEE000 symbolic feedback code.

output_GMT_seconds **(output)**
A 64-bit double floating-point number representing the current date and time in Greenwich, England, as the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). 19:00:01.078 on 16 May 1988 is second number 12,799,191,601.078. If GMT is not available from the system, *output_GMT_seconds* is set to 0 and CEEGMT terminates with a non-CEE000 symbolic feedback code.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E6	3	2502	The UTC/GMT was not available from the system.

Usage notes

- CEEDATE converts *output_GMT_Lilian* to a character date, and CEEDATM converts *output_GMT_seconds* to a character timestamp.
- CICS consideration—CEEGMT does not use the OS TIME macro.
- z/OS UNIX consideration—In multithread applications, CEEGMT affects only the calling thread.
- Setting the TOD (time-of-day) clock to anything before January 1, 1972 may produce unpredictable results in your applications.

For more information

- See "CEEGMTO—Get offset from Greenwich Mean Time to local time" on page 295 for more information about the CEEGMTO callable service.

Examples

1. Following is an example of CEEGMT called by C/C++.

```
/*Module/File Name: EDCGMT    */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _INT4     lilGMT_date;
  _FLOAT8   secGMT_date;

  CEEGMT(&lilGMT_date,&secGMT_date,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEGMT failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  printf("The current Lilian date in Greenwich,");
  printf(" England is %d\n", lilGMT_date);
}
```

2. Following is an example of CEEGMT called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTGMT
      *************************************************
      **                                             **
      ** IGZTGMT - Call CEEGMT to get current        **
      **          Greenwich Mean Time                **
      **                                             **
      ** In this example, a call is made to CEEGMT   **
      ** to return the current GMT as a Lilian date  **
      ** and as Lilian seconds. The results are      **
      ** displayed.                                  **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTGMT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  LILIAN                PIC S9(9) BINARY.
       01  SECS                  COMP-2.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLGMT.
           CALL "CEEGMT" USING LILIAN , SECS , FC.

           IF CEE000 of FC  THEN
               DISPLAY "The current GMT is also "
                   "known as Lilian day: " LILIAN
           DISPLAY "The current GMT in Lilian "
               "seconds is: " SECS
       ELSE
           DISPLAY "CEEGMT failed with msg "
               Msg-No of FC UPON CONSOLE
           STOP RUN
       END-IF.

       GOBACK.
```

3. Following is an example of CEEGMT called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMGMT                         */
 /****************************************************/
 /**                                                **/
 /** Function: CEEGMT - get current Greenwich Mean  **/
 /**           Time                                 **/
 /** In this example, CEEGMT is called to return    **/
 /** the current Greenwich Mean Time as the number  **/
 /** of days and number of seconds since            **/
 /** 14 October 1582. The Lilian date is then       **/
 /**  printed.                                      **/
 /**                                                **/
 /****************************************************/

 PLICGMT: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL LILIAN   REAL FIXED BINARY(31,0);
    DCL SECONDS  REAL FLOAT DECIMAL(16);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    /* Call CEEGMT to return current GMT as a        */
    /* Lilian date and Lilian seconds                */
    CALL CEEGMT ( LILIAN, SECONDS, FC );

    /* If CEEGMT ran successfully, print results     */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( LILIAN ||
          ' days have passed since 14 October 1582.' );
       END;
    ELSE  DO;
       DISPLAY( 'CEEGMT  failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLICGMT;
```

CEEGMTO—Get offset from Greenwich Mean Time to local time

CEEGMTO returns values to the calling routine representing the difference between the local system time and Greenwich Mean Time (GMT).

Syntax

```
►►──CEEGMTO──(──offset_hours──,──offset_minutes──,──offset_seconds──,──fc──────►

►──)──────────────────────────────────────────────────────────────────────────►◄
```

offset_hours **(output)**
A 32-bit binary integer representing the offset from GMT to local time, in hours. For example, for Pacific Standard Time, *offset_hours* equals -8. If local time offset is not available, *offset_hours* equals 0 and CEEGMTO terminates with a non-CEE000 symbolic feedback code.

offset_minutes **(output)**
A 32-bit binary integer representing the number of additional minutes that local time is ahead of or behind GMT. The range of *offset_minutes* is 0 to 59. If the local time offset is not available, *offset_minutes* equals 0 and CEEGMTO terminates with a non-CEE000 symbolic feedback code.

offset_seconds **(output)**
A 64-bit double floating-point number representing the offset from GMT to local time, in seconds. For example, Pacific Standard Time is eight hours behind GMT. If local time is in the Pacific time zone during standard time, CEEGMTO would return -28,800 (-8 * 60 * 60). *offset_seconds* can be used with CEEGMT to calculate local date and time. See "CEEGMT—Get current Greenwich Mean Time" on page 292 for more information.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E7	3	2503	The offset from UTC/GMT to local time was not available from the system.

Usage notes

- CEEDATM is used to convert number of seconds to a character timestamp.
- CICS consideration—CEEGMTO does not use the OS TIME macro.
- z/OS UNIX consideration—In multithread applications, CEEGMTO affects only the calling thread.

For more information

- See "CEEDATM—Convert seconds to character timestamp" on page 236 for more information about the CEEDATM callable service.

Examples

1. An example of CEEGMTO called by C/C++:

```
/*Module/File Name: EDCGMTO   */

#include <string.h>
```

```
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _INT4     GMT_hours,GMT_mins;
  _FLOAT8   GMT_secs;

  CEEGMTO(&GMT_hours,&GMT_mins,&GMT_secs,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEGMTO failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  printf("The difference between GMT and the local ");
  printf("time is:\n");
  printf("%d hours, %d minutes\n",GMT_hours,GMT_mins);
}
```

2. An example of CEEGMTO called by COBOL:

```
CBL LIB,QUOTE
      *Module/File Name: IGZTGMTO
      *************************************************
      **                                             **
      ** IGZTGMTO - Call CEEGMTO to get offset from  **
      **            Greenwich Mean Time to local     **
      **            time                             **
      **                                             **
      ** In this example, a call is made to CEEGMTO  **
      ** to return the offset from GMT to local time **
      ** as separate binary integers representing    **
      ** offset hours, minutes, and seconds. The     **
      ** results are displayed.                      **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTGMTO.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  HOURS                   PIC S9(9) BINARY.
       01  MINUTES                 PIC S9(9) BINARY.
       01  SECONDS COMP-2.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY. PROCEDURE DIVISION.
       PARA-CBLGMTO.
           CALL "CEEGMTO" USING HOURS , MINUTES ,
               SECONDS , FC.

           IF CEE000 of FC  THEN
               DISPLAY "Local time differs from GMT "
                   "by: " HOURS " hours, "
                   MINUTES " minutes, and "
```

```
                SECONDS " seconds. "
        ELSE
            DISPLAY "CEEGMTO failed with msg "
                Msg-No of FC UPON CONSOLE
            STOP RUN
        END-IF.

        GOBACK.
```

3. An example of CEEGMTO called by PL/I:

```
*PROCESS MACRO;
 /* Module/File Name: IBMGMTO                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEEGMTO - get the offset from        **/
 /**                    Greenwich Mean Time         **/
 /**                    to local time               **/
 /**                                                **/
 /****************************************************/
 PLIGMTO: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL HOURS REAL FIXED BINARY(31,0);
    DCL MINUTES REAL FIXED BINARY(31,0);
    DCL SECONDS REAL FLOAT DECIMAL(16);
    DCL 01 FC,                      /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

   /* Call CEEGMTO to return hours, minutes, and     */
   /* seconds that local time is offset from GMT     */

    CALL CEEGMTO ( HOURS, MINUTES, SECONDS, FC );

   /* If CEEGMTO ran successfully, print results     */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP EDIT('The difference between GMT and '
            || 'local time is ', HOURS, ':', MINUTES )
          (A, P'S99', A, P'99' );
       END;
    ELSE  DO;
       DISPLAY( 'CEEGMTO failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLIGMTO;
```

CEEGPID—Retrieve the Language Environment version and platform ID

CEEGPID retrieves the Language Environment version ID and the platform ID of the version and platform of Language Environment that is processing the currently active condition.

Syntax

```
►►──CEEGPID──(──CEE_Version_ID──,──Plat_ID──,──fc──)─────────────────────►◄
```

CEE_Version_ID **(output)**

A four-byte hexadecimal number representing the version of Language Environment that created this data block.

```
|pp|vv|rr|mm|
 pp = Product Number
 vv = Version
 rr = Release
 mm = Modification
```

The current value of this parameter is:

X'04010800'
Version 1 Release 8 Modification 0

X'04010900'
Version 1 Release 9 Modification 0

X'04010A00'
Version 1 Release 10 Modification 0

X'04010B00'
Version 1 Release 11 Modification 0

X'04010C00'
Version 1 Release 12 Modification 0

X'04010D00'
Version 1 Release 13 Modification 0

X'04020100'
Version 2 Release 1 Modification 0

Plat_ID **(output)**

A fullword integer representing the platform used for processing the current condition. The current values of this parameter are:

3	z/OS or VM
4	AS/400®

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Usage notes

- z/OS UNIX consideration—In multithread applications, CEEGPID affects only the calling thread.

- To make a decision concerning Language Environment and z/OS levels at assembly time instead of runtime, use CEEGLOB instead. See *z/OS Language Environment Programming Guide* for more information.

Examples

1. Following is an example of CEEGPID called by C/C++.

```
/*Module/File Name: EDCGPID   */
/****************************************************/
/****************************************************/
/* Note that the format of data returned by CEEGPID */
/* changed in OS/390 V2R10.  This sample tests the  */
/* version and chooses the appropriate format.      */
/****************************************************/
 #include <stdio.h>
 #include <string.h>
 #include <stdlib.h>
 #include <leawi.h>
 #include <ceeedcct.h>
int main(void) {
  _INT4 cee_ver_id, plat_id;
  _FEEDBACK fc;
  int Vmask= 0x00FF0000;
  int Rmask= 0x0000FF00;
  int Mmask= 0x000000FF;
  /* get the LE version and the platform id */
  CEEGPID(cee_ver_id,plat_id,fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEGPID failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
  /* If platform is z/OS and LE is at release 2.10 or later, */
  /* use the new interface definition.                        */
  if ((plat_id == 3) & (cee_ver_id > 290)) {
      printf("The LE version id is %08X\n",cee_ver_id);
      printf("             Version:   %d\n",(Vmask & cee_ver_id)>>16);
      printf("             Release:    %d\n",(Rmask & cee_ver_id)>>8);
      printf("        Modification:       %d\n\n",Mmask & cee_ver_id);
      }
   /* else use the old interface */
   else {
      printf("The LE version is %d\n",cee_ver_id);
  }
  printf("The current platform is ");
  switch(plat_id) {
     case 3: printf("z/OS\n");
             break;
     case 4: printf("AS/400\n");
             break;
    default: printf("unrecognized platform id\n");

  }
}
```

2. Following is an example of CEEGPID called by COBOL.

```
CBL LIB,QUOTE
      *************************************************
      *Module/File Name: IGZTGPID
      *************************************************
      **                                             **
      ** IGZTGPID - Call CEEGPID to retrieve the     **
      **            LE version and platform ID       **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
```

```
 PROGRAM-ID. IGZTGPID.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  VERSION                PIC S9(9) BINARY.
 01  VER-MAP REDEFINES VERSION.
     03 FILLER              PIC X(1).
     03 VER-VERSION         PIC X(1).
     03 VER-RELEASE         PIC X(1).
     03 VER-MOD             PIC X(1).
 01  WORK-AREA.
     03 WORK-BIN            PIC S9(4) BINARY.
     03 WORK-BIN-BY-BYTE REDEFINES WORK-BIN.

        05 WORK-BIN-BYTE1   PIC X(1).
        05 WORK-BIN-BYTE2   PIC X(1).
 01  VERSION-F              PIC 99.
 01  RELEASE-F              PIC 99.
 01  MOD-F                  PIC 99.
 01  PLATID                 PIC S9(9) BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity   PIC S9(4) BINARY.
             04  Msg-No     PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl   PIC X.
         03  Facility-ID    PIC XXX.
     02  I-S-Info           PIC S9(9) BINARY.
 PROCEDURE DIVISION.
 PARA-CBLGPID.
** Call CEEGPID to return the version and
**     platform ID
     CALL "CEEGPID" USING VERSION , PLATID , FC.
     IF  NOT CEE000 of FC  THEN
         DISPLAY "CEEGPID failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

** If platform is OS/390 or VM and version is greater than 290,
** then use different format of VERSION.
     IF PLATID = 3 AND
        VERSION > 290 THEN
** Format the version, release, and modification level
       MOVE 0 to WORK-BIN
       MOVE VER-VERSION TO WORK-BIN-BYTE2
       MOVE WORK-BIN TO VERSION-F
       MOVE VER-RELEASE TO WORK-BIN-BYTE2
       MOVE WORK-BIN TO RELEASE-F
       MOVE VER-MOD TO WORK-BIN-BYTE2
       MOVE WORK-BIN TO MOD-F
       DISPLAY "Currently running version " VERSION-F
                " release " RELEASE-F " modification " MOD-F
            " of IBM Language Environment"
     ELSE
       DISPLAY "Currently running version " VERSION
            " of IBM Language Environment"
     END-IF

** Evaluate PLATID to display this platform
     EVALUATE PLATID
       WHEN 3
         DISPLAY "on OS/390 or VM"
```

```
             WHEN 4
               DISPLAY "on an AS/400"
           END-EVALUATE

           GOBACK.
```

3. Following is an example of CEEGPID called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMGPID                          */
 /****************************************************/
 /**                                                **/
 /** Function: CEEGPID - Get LE/370 Version         **/
 /** and Platform ID                                **/
 /**                                                **/
 /** This example calls CEEGPID to get the          **/
 /** version and platform of Language               **/
 /** Environment that is currently running.         **/
 /** This information is then printed out.          **/
 /**                                                **/
 /****************************************************/
 PLIGPID: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL VERSION  REAL FIXED BINARY(31,0);
    DCL PLATID   REAL FIXED BINARY(31,0);
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    /* Call CEEGPID to get the version and platform  */
    /*    of Language Environment that is currently  */
    /*    running                                    */
    CALL CEEGPID ( VERSION, PLATID, FC );

    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST
          ('Language Environment Version ' || VERSION);
       PUT LIST (' is running on system ');
       SELECT (PLATID);
          WHEN (2) PUT LIST( 'OS/2');
          WHEN (3) PUT LIST( 'MVS/VM/370');
          WHEN (4) PUT LIST( 'AS/400');
          END /* Case of PLATID */;
       END;
    ELSE  DO;
       DISPLAY( 'CEEGPID failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLIGPID;
```

CEEGQDT—Retrieve q_data_token

CEEGQDT retrieves the *q_data_token* from the Instance-Specific Information (ISI). CEEGQDT is particularly useful when you have user-written condition handlers registered by CEEHDLR.

Syntax

```
►►──CEEGQDT──(──cond_rep──,──q_data_token──,──fc──)──────────────────────────►◄
```

`cond_rep` **(input)**
A condition token defining the condition for which the *q_data_token* is retrieved.

`q_data_token` **(output)**
A pointer to the q_data associated with condition token *cond_rep*.

`fc` **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service.

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0EE	3	0462	Instance-specific information for the condition token with message number *message-number* and facility ID *facility-id* could not be found.
CEE0EG	3	0464	Instance-specific information for the condition token with message number *message-number* and facility ID *facility-id* did not exist.

Usage notes

- z/OS UNIX consideration—In multithread applications, CEEGQDT affects only the calling thread.

For more information

- For more information about the various types of q_data structures, see *z/OS Language Environment Programming Guide*.
- For more information about the CEEHDLR callable service, see "CEEHDLR—Register user-written condition handler" on page 317.
- For more information about the CEESGL callable service, see "CEESGL—Signal a condition" on page 415.

Examples

1. Following is an example of CEEGQDT called by C/C++.

```
/*Module/File Name: EDCGQDT   */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <leawi.h>
```

```
#include <ceeedcct.h>
#ifdef __cplusplus
  extern "C" {
#endif
  void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
  }
#endif

typedef struct {       /* condition info structure */
   int   error_value;
   char err_msg_80¨;
   int   retcode;
} info_struct;
int main(void) {
   _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;
  info_struct *info;
 /* .
    .
    . */
  /* register the condition handler */
  token = 99;
  routine.address = (_POINTER)&handler;;
  routine.nesting = NULL;
  CEEHDLR(&routine,&token,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEHDLR failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
 /* .
    .
    . */
  /* set up the condition info structure */
  info = (info_struct *)malloc(sizeof(info_struct));
  if (info == NULL) {
     printf("error allocating info_struct\n");
     exit(2399);
  }

  info->error_value = 86;
  strcpy(info->err_msg,"Test message");
  info->retcode = 99;
  /* set qdata to be the condition info structure */
  qdata = (int)info;
  /* build the condition token */
  c_1 = 3;
  c_2 = 99;
  cond_case = 1;
  sev = 3;
  control = 0;
  memcpy(facid,"ZZZ",3);
  isi = 0;
  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&condtok,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  /* signal the condition */
  CEESGL(&condtok,&qdata,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEESGL failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
 /* .
    .
    . */
}
void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {
   _FEEDBACK qdatafc;
   _INT4 idata;
   info_struct *qdata;
 /* .
    .
    . */
   /* get the q_data_token from the ISI */
   CEEGQDT(fc, &idata, &qdatafc);
   if ( _FBCHECK ( qdatafc , CEE000 ) != 0 ) {
      printf("CEEGQDT failed with message number %d\n",
             qdatafc.tok_msgno);
      *result = 20; /* percolate */
      return;
   }
   /***********************************************/
   /* set info_struct pointer to address return by */
   /* CEEGQDT                                      */
   /***********************************************/
   qdata = (info_struct *) idata;
   /* use the condition info structure (qdata) */
   if (qdata->error_value == 86) {
      printf("%.12s\n",qdata->err_msg);
      printf("retcode = %d\n",qdata->retcode);
      *result = 10; /* resume this is what we want */
      return;
   }
 /* .
    .
    . */
   *result = 20; /* percolate */
}
```

2. Following is an example of CEEGQDT called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTGQDT
      *************************************************
      ** DRVGQDT - Drive sample program for CEEGQDT  **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRVGQDT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE                PROCEDURE-POINTER.
       01  TOKEN                  PIC S9(9) BINARY.
       01  SEV                    PIC S9(4) BINARY.
       01  MSGNO                  PIC S9(4) BINARY.
       01  CASE                   PIC S9(4) BINARY.
       01  SEV2                   PIC S9(4) BINARY.
       01  CNTRL                  PIC S9(4) BINARY.
       01  FACID                  PIC X(3).
       01  ISINFO                 PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
```

```
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    01  QDATA                   PIC S9(9) BINARY.
    01  CONDTOK.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.

   PROCEDURE DIVISION.
   ** Register handler
        SET ROUTINE TO ENTRY "CBLGQDT".
        CALL "CEEHDLR" USING ROUTINE , TOKEN , FC.
        IF  NOT CEE000 of FC  THEN
            DISPLAY "CEEHDLR failed with msg "
                Msg-No of FC UPON CONSOLE
            STOP RUN
        END-IF.
   ** Signal a condition
        MOVE 1 TO QDATA.
        SET CEE001 of CONDTOK to TRUE.
        MOVE ZERO to I-S-Info of CONDTOK.
        CALL "CEESGL" USING CONDTOK , QDATA , FC.
        IF CEE000 of FC  THEN
            DISPLAY "**** Resumed execution in the "
              "routine which registered the handler"
        ELSE
            DISPLAY "CEESGL failed with msg "
                Msg-No of FC UPON CONSOLE
        END-IF.
   ** UNregister handler
        CALL "CEEHDLU" USING ROUTINE , TOKEN , FC.
        IF NOT CEE000 of FC  THEN
            DISPLAY "CEEHDLU failed with msg "
                Msg-No of FC UPON CONSOLE
        END-IF.
        STOP RUN.
    END PROGRAM DRVGQDT.
   *******************************************
   ** CBLGQDT - Call CEEGQDT to get        **
   **          the Q_DATA_TOKEN            **
   *******************************************
    IDENTIFICATION DIVISION.
    PROGRAM-ID. CBLGQDT.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  FC.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
```

```
                   REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    01  QDATA                   PIC S9(9) BINARY.
    LINKAGE SECTION.
    01   CURCOND.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.       01  TOKEN
            PIC S9(9) BINARY.
    01  RESULT          PIC S9(9) BINARY.
        88  RESUME          VALUE 10.
    01  NEWCOND.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    PROCEDURE DIVISION
        USING CURCOND, TOKEN, RESULT, NEWCOND.
    PARA-CBLGQDT.
   ** Obtain the Qdata for the current condition
        CALL "CEEGQDT" USING CURCOND , QDATA , FC.
        IF  CEE000 of FC  THEN
            DISPLAY "QDATA for " Facility-ID of
                CURCOND Msg-No of CURCOND
                " is " QDATA
        ELSE
            DISPLAY "CEEGQDT failed with msg "
                Msg-No of FC UPON CONSOLE
        END-IF.
         SET RESUME TO TRUE.
        GOBACK.
    END PROGRAM CBLGQDT.
```

3. The following example uses a COBOL program and handler to establish the condition handling environment prior to calling a PL/I subroutine to illustrate the use of the callable service from PL/I.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTGQDP
      *************************************************
      ** IGZTGQDP - Drive sample program for CEEGQDT **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTGQDP.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE                 PROCEDURE-POINTER.
```

```
 01  TOKEN                   PIC S9(9) BINARY.
 01  SEV                     PIC S9(4) BINARY.
 01  MSGNO                   PIC S9(4) BINARY.
 01  CASE                    PIC S9(4) BINARY.
 01  SEV2                    PIC S9(4) BINARY.
 01  CNTRL                   PIC S9(4) BINARY.
 01  FACID                   PIC X(3).
 01  ISINFO                  PIC S9(9) BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 01  QDATA                   PIC S9(9) BINARY.
 01  CONDTOK.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION.
** Register handler
     SET ROUTINE TO ENTRY "HDLGQDT".
     CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
     IF  NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLR failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
** Signal a condition
     MOVE 1 TO QDATA.
     SET CEE001 of CONDTOK to TRUE.
     MOVE ZERO to I-S-Info of CONDTOK.
     CALL "CEESGL" USING CONDTOK, QDATA, FC.
     IF CEE000 of FC  THEN
         DISPLAY "**** Resumed execution in the "
           "routine which registered the handler"
     ELSE
         DISPLAY "CEESGL failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.
** UNregister handler
     CALL "CEEHDLU" USING ROUTINE, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLU failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.
     STOP RUN.
 END PROGRAM IGZTGQDP.

*************************************************
** HDLGQDT -- COBOL condition handler to call  **
```

```
      **            PL/I routine for actual work.    **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. HDLGQDT.
       DATA DIVISION.
       LINKAGE SECTION.
       01   CURCOND.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       01  TOKEN                   PIC S9(9) BINARY.
       01  RESULT                  PIC S9(9) BINARY.
       01  NEWCOND.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                        REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION
           USING CURCOND, TOKEN, RESULT, NEWCOND.
       PARA-CBLGQDT.

       ** Invoke the PL/I routine to handle condition

           CALL "IBMGQDT"
               USING ADDRESS OF CURCOND,
                     ADDRESS OF TOKEN,
                     ADDRESS OF RESULT,
                     ADDRESS OF NEWCOND.
           GOBACK.

       END PROGRAM HDLGQDT.
```

4. Following is an example of CEEGQDT called by COBOL.

```
*PROCESS OPT(0), MACRO;
 /* Module/File Name: IBMGQDT                        */
 /***************************************************/
 /**                                               **/
 /** Function:  CEEGQDT -- get qualifying data      **/
 /**                                               **/
 /***************************************************/
 IBMGQDT: PROC (@CONDTOK, @TOKEN, @RESULT, @NEWCOND)
             OPTIONS(COBOL);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    /* Parameters */
    DCL  @CONDTOK    POINTER;
    DCL  @TOKEN      POINTER;
    DCL  @RESULT     POINTER;
```

```
   DCL  @NEWCOND    POINTER;
   DCL  01 CONDTOK  BASED(@CONDTOK),
                               /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),     /* Facility ID */
           03 ISI  /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
   DCL  TOKEN     BASED(@TOKEN)  REAL FIXED BIN(31,0);
   DCL  RESULT    BASED(@RESULT) REAL FIXED BIN(31,0);

   DCL  01 NEWCOND      BASED(@NEWCOND),
                               /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),     /* Facility ID */
           03 ISI  /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
   /* Local identifiers */
   DCL  QDATA       REAL FIXED BINARY(31,0);
   DCL  01 FC,                 /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),     /* Facility ID */
           03 ISI  /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

   IF FBCHECK(CONDTOK, CEE001) THEN /* expected */ DO;

      /* Call CEEGQDT with condition token defined  */
      /*    above to retrieve associated q_data     */
      CALL CEEGQDT ( CONDTOK, QDATA, FC );
      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST( 'Qualifying data for current '
            || ' condition is ' || QDATA );
         RESULT = 10 /* Resume */;
         END;
      ELSE  DO;
         DISPLAY('CEEGQDT failed with msg ' ||
                                         FC.MsgNo);
         NEWCOND = FC;
         RESULT = 30 /* Promote */;
         END;
      END;
   ELSE /* Unexpected condition -- percolate */ DO;
      DISPLAY( 'User condition handler entered for '
         || CONDTOK.FacID || ' condition...');
      DISPLAY( '... with message number ' ||
                                    CONDTOK.MsgNo);
      RESULT = 20 /* Percolate */;
      END;

   RETURN;
END IBMGQDT;
```

CEEGTJS—Retrieves the value of an exported JCL symbol

CEEGTJS retrieves and returns to the caller the symbol value and length of the requested exported JCL symbol.

Syntax

```
►►──CEEGTJS──(──function_code──,──symbol_name──,──symbol_value──,──────────────►

►─value_length──,──fc──)──────────────────────────────────────────────────────►◄
```

function_code **(input)**
A fullword integer that contains the function code of the following value:

1 Retrieves the value and its associated length of an exported JCL symbol.

symbol_name **(input)**
A halfword length-prefixed character string (VSTRING), representing the name of an exported JCL symbol to be retrieved.

symbol_value **(output)**
A 255-byte fixed-length string. On return from this service, the s*symbol_value* contains the value of the exported JCL symbol. If the length of the exported JCL symbol is shorter than 255 characters, the returned string is padded with blanks.

value_length **(output)**
A fullword integer that contains the length of the value of the specified JCL symbol.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3L9	0	3753	The input symbol cannot be found in the current job step.
CEE3LA	3	3754	Incorrect parameters detected.
CEE3QS	1	3932	The system service *service* failed with return code *return_code* and reason code *reason_code*.

Usage notes

- Lowercase characters in the *symbol_name* are converted to uppercase by CEEGTJS.
- For more information about JCL symbols, refer to the topic on system symbols and JCL symbols in *z/OS MVS JCL Reference*.

Examples

1. This example uses CEEGTJS to retrieve the value of an exported JCL symbol.

```
/*Module/File Name: EDCGTJS    */
/*******************************************************************/
/*                                                                 */
/* THIS EXAMPLE CALLS CEEGTJS TO RETRIEVE THE VALUE OF AN EXPORTED   */
/* JCL SYMBOL.                                                     */
/*                                                                 */
/*******************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
   _FEEDBACK fc;
   _INT4 funcode;
   _CHAR255 symvalue;
   _VSTRING symname;
   _INT4 valuelen;
   char *symbol="SYM1";

   /* Setting the function code */
   funcode=1;

 /* Preparing the JCL symbol */
   symname.length=strlen(symbol);
   memcpy(symname.string, symbol,strlen(symbol));

/* Retrieving the value of the JCL symbol */
   CEEGTJS(&funcode,&symname,symvalue,&valuelen,&fc);
   if( _FBCHECK (fc, CEE000) !=0) {
    printf("CEEGTJS failed with message number %d\n",
         fc.tok_msgno);
    exit(1);
  }
   symvalue[valuelen]='\0';
   printf("The value of JCL symbol %s is %s. The length
          of the value is %d\n",symbol,symvalue,valuelen);
}
```

Use the following JCL to run EDCGTJS:

```
//JOB1      JOB    FELE,MSGLEVEL=(2,0)
//STEP1     EXEC   PGM=EDCGTJS
//E1        EXPORT SYMLIST=(SYM1,SYM2,SYM3)
//S1        SET    SYM1=XXXX
//S2        SET    SYM2=YYYY
//STEPLIB   DD     DSN=USER.LOADLIB,DISP=SHR
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
```

Running this example would produce the following output:

```
The value of JCL symbol SYM1 is XXXX. The length of the value is 4.
```

CEEGTST—Get heap storage

CEEGTST gets storage from a heap whose ID you specify. It is used to acquire both large and small blocks of storage. CEEGTST always allocates storage that is addressable by the caller. Therefore, if the caller is in 24-bit addressing mode, or if HEAP(,,BELOW) is in effect, the storage returned is always below the 16M line. Above-the-line storage is returned only if the caller is in 31-bit addressing mode and HEAP(,,ANY) is in effect.

All requests for storage are conditional. If storage is not available, the feedback code (*fc*) is set and returned to you, but the thread does **not** abend. When storage is not available, the appropriate action in the member environment should be taken. One option is to use the CEESGL callable service to signal the Language Environment condition handler with the returned feedback code.

Storage obtained by CEEGTST can be freed by a call to CEEFRST or CEEDSHP. You can also free storage by using a language intrinsic function. If storage is not explicitly freed, it is freed automatically at termination.

If you have specified a *heap_alloc_value* in the STORAGE runtime option, all storage allocated by CEEGTST is initialized to *heap_alloc_value*. Otherwise, it is left uninitialized.

If the value specified in the *size* parameter of CEEGTST is greater than the size of an increment (as specified in the HEAP runtime option), all of the requested storage (rounded up to the nearest doubleword) is allocated in a single system-level call.

Heap storage is acquired by a system-level get storage call in increments of *init_size* and *incr_size* bytes as specified by the HEAP runtime option, or in the CEECRHP callable service. If the increment size is chosen appropriately, only a few of the calls to CEEGTST result in a system call. The storage report generated when the RPTSTG runtime option is specified shows the number of system-level get storage calls required. This helps you tune the *init_size* and *incr_size* fields in order to minimize calls to the operating system.

Syntax

```
►►──CEEGTST──(──heap_id──,──size──,──address──,──fc──)──────────────────────►◄
```

heap_id **(input)**
A fullword binary signed integer. *heap_id* is a token denoting the heap in which the storage is allocated. A *heap_id* of 0 allocates storage from the initial heap (or user heap). Any other *heap_id* must be a value obtained from the CEECRHP callable service. If the *heap_id* you specify is invalid, no storage is allocated. CEEGTST terminates with a non-CEE000 symbolic feedback code and the value of the *address* parameter is undefined.

size **(input)**
A fullword binary signed integer. *size* represents the amount of storage allocated, in bytes. If the specified amount of storage cannot be obtained, no storage is allocated, CEEGTST terminates with a non-CEE000 symbolic feedback code, and the value of the *address* parameter is undefined.

`address` **(output)**

A fullword address pointer. *address* is the main storage address of the first byte of allocated storage. If storage cannot be obtained, *address* remains undefined. Storage is always allocated on a doubleword boundary. This parameter contains an address that is returned as output to CEECZST.

`fc` **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0P2	4	0802	Heap storage control information was damaged.
CEE0P3	3	0803	The heap identifier in a get storage request or a discard heap request was unrecognized.
CEE0P8	3	0808	Storage size in a get storage request (CEEGTST) or a reallocate request (CEECZST) was not a positive number.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage (CEECZST) request.

Usage notes

- COBOL considerations— If you want to use the CEEGTST callable service in a 24-bit addressing mode COBOL program to request 24-bit heap storage, you can make a static call to CEEGTST without any additional changes. If you want to call it dynamically (either by using the COBOL DYNAM compiler option, or using the "CALL identifier" statement, where identifier is a variable that holds the name of the program you want to call), you must also specify HEAP(,,BELOW,,,) as a runtime option.

  This runtime specification affects all programs in the enclave using the user heap, not just the 24-bit addressing mode program. If this is undesirable, one alternative is to use the CEECRHP callable service to create an additional heap, specifying for the options parameter a value that will create the heap BELOW. The 24-bit addressing mode COBOL program can then obtain storage from this additional heap using the CEEGTST callable service.
- PL/I considerations—Storage allocated within PL/I AREAs is managed by PL/I. Therefore, only PL/I language functions can allocate and free storage within a PL/I area.
- Based upon the layout of a PL/I structure, PL/I might adjust the starting byte of the PL/I structure to a non-doubleword aligned byte. The difference between the doubleword boundary and the first byte of such a structure is known as the *hang*. Because Language Environment callable storage services do not adjust the starting byte, you must be careful using callable services to allocate storage for PL/I structures. Use either the PL/I ALLOCATE statement or fully defined structures and aggregates to avoid this problem.
- CICS considerations—In a CICS environment, *size* should not exceed 1024M (1 gigabyte or X'40000000') when running in AMODE ANY, and 65,504 bytes when running in 24-bit addressing mode. These CICS restrictions are subject to change from one release of CICS to another. Portable applications should respect current CICS limitations.

- z/OS UNIX considerations—CEEGTST applies to the enclave. Any thread can free the allocated storage.

For more information

- See "HEAP" on page 36 for further information about the Language Environment HEAP runtime option.
- For more information about the CEESGL callable service, see "CEESGL—Signal a condition" on page 415.
- For more information about the CEEFRST callable service, see "CEEFRST—Free heap storage" on page 282.
- For more information about the CEEDSHP callable service, see "CEEDSHP—Discard heap" on page 256.
- For more information about the STORAGE runtime option, see "STORAGE" on page 79.
- For more information about the CEECRHP callable service, see "CEECRHP—Create new additional heap" on page 221.
- For more information about the RPTSTG runtime option, see "RPTSTG" on page 71.
- For more information about the CEECRHP callable service, see "CEECRHP—Create new additional heap" on page 221.
- For more information about the CEECZST callable service, see "CEECZST—Reallocate (change size of) storage" on page 225.

Examples

1. Following is an example of CEEGTST called by C/C++.

```
/*Module/File Name: EDCGTST   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

   _INT4 heapid, size;
   _POINTER address;
   _FEEDBACK fc;
  /* .
     .
     . */
   heapid = 0;    /* get storage from initial heap   */
   size = 4000;   /* number of bytes of heap storage */

   /* obtain the storage using CEEGTST */
   CEEGTST(&heapid,&size,&address,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEGTST failed with message number %d\n",
             fc.tok_msgno);
      exit(99);
   }
  /* .
     .
     . */
   /* free the storage that was previously obtained */
   /* using CEEGTST */
   CEEFRST(&address,&fc);

   /* check the first 4 bytes of the feedback token */
   /* (0 if successful) */
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEFRST failed with message number %d\n",
             fc.tok_msgno);
      exit(99);
   }
  /* .
     .
     . */
}
```

2. Following is an example of CEEGTST called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTGTST
      ****************************************************
      **                                                **
      ** IGZTGTST - Call CEEGTST to get heap storage   **
      **                                                **
      ** In this example, a call is made to CEEGTST to **
      ** obtain 4000 bytes of storage from the initial **
      ** heap (HEAPID=0).                               **
      **                                                **
      ****************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTGTST.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  HEAPID                  PIC S9(9) BINARY.
       01  STGSIZE                 PIC S9(9) BINARY.
       01  ADDRSS                  POINTER.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
      **************************************************
      ** Specify 0 to get storage from initial heap. **
      ** Specify 4000 to get 4000 bytes of storage.  **
      ** Call CEEGTST to obtain storage.             **
      **************************************************
       PARA-CBLGTST.
           MOVE 0 TO HEAPID.
           MOVE 4000 TO STGSIZE.

           CALL "CEEGTST" USING HEAPID, STGSIZE,
                                ADDRSS, FC.
           IF CEE000 of FC  THEN
               DISPLAY "Obtained " STGSIZE " bytes of"
                   " storage at location " ADDRSS
                   " from heap number " HEAPID
           ELSE
               DISPLAY "CEEGTST failed with msg "
                   Msg-No of FC UPON CONSOLE
```

```
              STOP RUN
          END-IF.

          GOBACK.
```

3. Following is an example of CEEGTST called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMGTST                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEEGTST - Get Heap Storage           **/
 /**                                                **/
 /** In this example, a call is made to CEEGTST to  **/
 /** request 4000 bytes of storage from the         **/
 /** initial heap.                                  **/
 /**                                                **/
 /****************************************************/
 PLIGTST: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL HEAPID   REAL FIXED BINARY(31,0);
    DCL STGSIZE  REAL FIXED BINARY(31,0);
    DCL ADDRSS   POINTER;
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    HEAPID = 0; /* get storage from the initial heap */
    STGSIZE = 4000;   /* get 4000 bytes of storage   */

    /* Call CEEGTST to obtain the storage            */
    CALL CEEGTST ( HEAPID, STGSIZE, ADDRSS, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'Obtained ' || STGSIZE
          || ' bytes of storage at location '
          || DECIMAL( UNSPEC( ADDRSS1 ) )
          || ' from heap ' || HEAPID );
       END;
    ELSE  DO;
       DISPLAY( 'CEEGTST failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLIGTST;
```

CEEHDLR—Register user-written condition handler

CEEHDLR registers a user-written condition handler for the current stack frame. The user condition handler is invoked when:

- It is registered for the current stack frame by CEEHDLR, and
- The Language Environment condition manager requests the condition handler associated with the current stack frame handle the condition.

Language Environment places the user-written condition handlers associated with each stack frame in a queue. The queue can be empty at any given time. The Language Environment condition manager invokes the registered condition handlers in LIFO (last in, first out) order to handle the condition.

The opposite of CEEHDLR, which registers a user-written condition handler, is CEEHDLU, which unregisters the handler. You do not necessarily need to use CEEHDLU to remove user-written condition handlers you registered with CEEHDLR. Any user-written condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by Language Environment, but only when the associated stack frame is removed from the stack.

Language Environment condition handlers are driven only for synchronous conditions.

Syntax

```
►►──CEEHDLR──(──routine──,──token──,──fc──)──────────────────────────────►◄
```

routine **(input)**
An entry variable or entry constant for the routine called to process the condition. The entry variable or constant must be passed by reference. The routine must be an external routine; that is, it must not be a nested routine.

token **(input)**
A fullword integer of information you want passed to your user handler each time it is called. This can be a pointer or any other fullword integer you want to pass.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE080	1	0256	The user-written condition handler routine specified was already registered for this stack frame. It was registered again.
CEE081	3	0257	The routine specified contained an invalid entry variable.

Usage notes

- PL/I MTF consideration—CEEHDLR is not supported in PL/I MTF applications. This includes any CEEHDLR service called from a COBOL program in the application.
- COBOL consideration—You should not call CEEHDLR from a nested COBOL program.
- z/OS UNIX consideration—In multithread applications, CEEHDLR affects only the calling thread.

- C consideration—The CEEHDLR service does not save Writeable Static Area (WSA) information about the user handler, so it's possible that the user handler will be given control with the wrong WSA. Specifically, the user handlers on the stack will be invoked with the WSA of the routine that incurred the condition. The preferred method of handling conditions from a C application is to use C signal handlers.

For more information

- For more information about the CEEHDLU callable service, see "CEEHDLU—Unregister user-written condition handler" on page 326.

Examples

1. Following is an example of CEEHDLR called by C/C++.

```
/*Module/File Name: EDCHDLR   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

#ifdef __cplusplus
  extern "C" {
#endif
  void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
  }
#endif

int main(void) {

  _FEEDBACK fc;
  _ENTRY routine;
  _INT4 token;

  /* set the routine structure to point to the handler */
  /* and use CEEHDLR to register the user handler      */

  token = 99;
  routine.address = (_POINTER)&handler;
  routine.nesting = NULL;

  CEEHDLR(&routine,&token,&fc);

  /* verify that CEEHDLR was successful */
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEHDLR failed with message number %d\n",
            fc.tok_msgno);
     exit (2999);
  }
 /*

⋮
*/
}
/*******************************************************/
/* handler is a user condition handler                 */
/*******************************************************/
void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {
 /*
⋮
 */
}
```

2. Following is an example of CEEHDLR called by a COBOL program that registers a handler routine.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTHDLR
      ***********************************************
      **                                           **
      ** CBLHDLR - Call CEEHDLR to register a user **
      **           condition handler               **
      **                                           **
      ***********************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLHDLR.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE  PROCEDURE-POINTER.
       01  TOKEN                  PIC S9(9) BINARY.
       01  SEV                    PIC S9(4) BINARY.
       01  MSGNO                  PIC S9(4) BINARY.
       01  CASE                   PIC S9(4) BINARY.
       01  SEV2                   PIC S9(4) BINARY.
       01  CNTRL                  PIC S9(4) BINARY.
       01  FACID                  PIC X(3).
       01  ISINFO                 PIC S9(9) BINARY.
       01  QDATA                  PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       01  CONDTOK.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLHDLR.
           SET ROUTINE TO ENTRY "HANDLER".
           CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY "CEEHDLR failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
      *  RAISE A SIGNAL

       PARA-CBLSGL.
```

```
*************************************************
** Call CEENCOD with the values assigned above **
** to build a condition token "CONDTOK"        **
** Set CONDTOK to sev=3, msgno=1 facid=CEE. We **
** raise a sev 3 to ensure our handler is driven*
*************************************************
     MOVE 3 TO SEV.
     MOVE 1 TO MSGNO.
     MOVE 1 TO CASE.
     MOVE 3 TO SEV2.
     MOVE 1 TO CNTRL.
     MOVE "CEE" TO FACID.
     MOVE 0 TO ISINFO.

     CALL "CEENCOD" USING SEV, MSGNO, CASE,
         SEV2, CNTRL, FACID, ISINFO, CONDTOK, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEENCOD failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

*************************************************
** Call CEESGL to signal the condition with    **
**     the condition token and qdata described **
**     in CONDTOK and QDATA                    **
*************************************************
     MOVE 0 TO QDATA.
     CALL "CEESGL" USING CONDTOK, QDATA, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEESGL  failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of a COBOL user-written condition handler that is registered by CBLHDLR and unregistered by CBLHDLU.

```
CBL LIB,QUOTE,NOOPT,NODYNAM
      *Module/File Name: IGZTHAND
      *************************************************
      **                                             **
      ** DRVHAND - Drive sample program for COBOL    **
      **           user-written condition handler.   **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRVHAND.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE            PROCEDURE-POINTER.
       01  DENOMINATOR        PIC S9(9) BINARY.
       01  NUMERATOR          PIC S9(9) BINARY.
       01  RATIO              PIC S9(9) BINARY.
       01  TOKEN              PIC S9(9) BINARY VALUE 0.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
```

```
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.

 REGISTER-HANDLER.
************************************************
** Register handler                           **
************************************************
     SET ROUTINE TO ENTRY "HANDLER".
     CALL "CEEHDLR" USING ROUTINE , TOKEN , FC.
     IF  NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLR failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

 RAISE-CONDITION.
************************************************
** Cause a zero-divide condition.             **
************************************************
     MOVE 0 TO DENOMINATOR.
     MOVE 1 TO NUMERATOR.
     DIVIDE NUMERATOR BY DENOMINATOR
            GIVING RATIO.
     DISPLAY "Execution continues following "
         "divide-by-zero exception".
 UNREGISTER-HANDLER.
************************************************
** UNregister handler                         **
************************************************
     CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLU failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.
     STOP RUN.
 END PROGRAM DRVHAND.

 IDENTIFICATION DIVISION.
 PROGRAM-ID. HANDLER.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 LINKAGE SECTION.
 01  TOKEN                  PIC S9(9) BINARY.
 01  RESULT                 PIC S9(9) BINARY.
     88 RESUME                  VALUE 10.
 01  CURCOND.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 01  NEWCOND.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
             03  Case-2-Condition-ID
                       REDEFINES Case-1-Condition-ID.
                 04  Class-Code  PIC S9(4) BINARY.
                 04  Cause-Code  PIC S9(4) BINARY.
             03  Case-Sev-Ctl    PIC X.
             03  Facility-ID     PIC XXX.
         02  I-S-Info            PIC S9(9) BINARY.
     PROCEDURE DIVISION USING CURCOND, TOKEN,
                              RESULT, NEWCOND.

     PARA-HANDLER.
         DISPLAY "Entered user handler for condition"
            " with message number " Msg-No Of CURCOND
            " -- will resume execution".
         SET RESUME TO TRUE.

         GOBACK.
     END PROGRAM HANDLER.
```

4. Following is an example of a PL/I program to handle divide-by-zero condition.

```
*Process macro;
 /* Module/File Name: IBMHDLR                            */
 /****************************************************/
 /*                                                      */
 /*  EXCOND              .-> DIVZERO                     */
 /*  - register handler   |   - force a divide-by-0  */
 /*  - call DIVZERO     --'                              */
 /*  ==> "resume point"                                  */
 /*  - unregister handler                                */
 /*                         USRHDLR:                     */
 /*                         - if divide-by-zero then */
 /*                           - move resume cursor   */
 /*                           - resume at "resume"   */
 /*                                          point   */
 /*                                                      */
 /****************************************************/
 Excond :Proc Options(Main);

   /**************************************************/
   /* Important elements are found in these includes */
   /* - feedback declaration                          */
   /* - fbcheck macro call                            */
   /* - condition tokens such as CEE000               */
   /* - entry declarations such as ceehdlr            */
   /**************************************************/

   %include ceeibmct;
   %include ceeibmaw;

   dcl Usrhdlr external entry;

   dcl 1 fback feedback;
   dcl divisor fixed bin(31);
   dcl token   fixed bin(31);

   /*********************************************/
   /* Register a user-written condition handler */
   /*********************************************/
   token = 97;
   Call ceehdlr(Usrhdlr, token, fback);
   If fbcheck (fback, cee000) then
     display ('MAIN: registered USRHDLR');
   else
     do;
       display ('CEEHDLR failed with message number ' ||
                fback.MsgNo);
       stop;
```

```
  end;

 /*********************************************/
 /* Call DIVZERO to divide by zero           */
 /*  and drive USRHDLR                       */
 /*********************************************/
 divisor = 0;
 call divzero (divisor);
 display ('MAIN: resumption after DIVZERO');
/*********************************************/
 /* Unregister the user condition handler    */
 /*********************************************/

 Call ceehdlu (Usrhdlr, fback);
 If fbcheck (fback, cee000) then
   display ('MAIN: unregistered USRHDLR');
 else
   do;
     display ('CEEHDLU failed with message number ' ||
               fback.MsgNo);
     stop;
   end;

   /*********************************************/
   /* Subroutine that simply raises ZERODIVIDE  */
   /*********************************************/
   divzero: proc (arg);
     dcl arg fixed bin(31);

     display('  DIVZERO: starting.');
     arg = 1 / arg;
     display('  DIVZERO: Returning to its caller');

   end divzero;

end Excond;
```

5. Following is an example of CEEHDLR called by PL/I.

```
AHDL     TITLE 'Main program that registers a handler'
*
*        Symbolic Register Definitions and Usage
*
R0       EQU   0             Parm list addr (CMS only)
R1       EQU   1             Parm list addr, 0=no parms
R10      EQU   10            Base reg for executable code
R12      EQU   12            Language Environment Common Anchor Area addr
R13      EQU   13            Dynamic Storage Area addr
R14      EQU   14            Return point addr
R15      EQU   15            Entry point address
*
*        Prologue
*
CEEHDLRA CEEENTRY AUTO=DSASIZ,    Main memory to obtain   *
               MAIN=YES,     This program is a MAIN prog  *
               PPA=PPA1,     Our Program Prolog Area      *
               BASE=R10      Base reg for executable code
         USING CEECAA,R12    Addressing for LE/370 CAA
         USING CEEDSA,R13    Addressing for dynamic data
*
*        Announce ourselves
*
         WTO   'CEEHDLRA Says "HELLO"',ROUTCDE=11
*
*        Register User Handler
*
         LA    R1,USRHDLPP   Get addr of proc-ptr to Hdlr
         ST    R1,PARM1      Make it 1st parameter
         LA    R1,TOKEN      Get addr of 32-bit token
         ST    R1,PARM2      Make it 2nd parameter
         LA    R1,FEEDBACK   Get addr of feedback code
         ST    R1,PARM3      Make it 3rd parameter
         LA    R1,HDLRPLST   Point to CEEHDLR's parm list
         CALL  CEEHDLR       Invoke CEEHDLR service
         CLC   FEEDBACK,=XL12'00' Check for success..
         BE    HDLRGOOD      Skip diagnostics if success
*                            Failure.. issue diagnostics
         WTO   '**** Call to CEEHDLR failed ****',       *
                ROUTCDE=11
         ABEND 1,DUMP        Terminate program with Dump
HDLRGOOD EQU   *             Handler registered OK
*  ... code covered by User-Written Handler goes here...
*        Un-Register User Handler
*
         LA    R1,USRHDLPP   Get addr of proc-ptr to Hdlr
         ST    R1,HDLUPRM1   Make it 1st parameter
         LA    R1,HDLUFBC    Address for feedback code
         ST    R1,HDLUPRM2   Make it 2nd parameter
         LA    R1,HDLUPLST   Point to CEEHDLU parm list
         CALL  CEEHDLU       Invoke CEEHDLU service
*        Bid a fond farewell
         WTO   'CEEHDLRA Says "GOOD-BYE"',ROUTCDE=11
*
*        Epilogue
*
         CEETERM RC=4,MODIFIER=1  Terminate program
*
*        Program Constants and Local Static Variables
*
USRHDLPP DC    V(USRHDLR),A(0)    Procedure-ptr to Handlr
*
         LTORG ,                  Place Literal Pool here
         EJECT
PPA1     CEEPPA ,            Our Program Prolog Area
         EJECT
         CEEDSA ,            Map CEE Dynamic Save Area
*
*        Local Automatic (Dynamic) Storage.
*
HDLRPLST DS    0F
PARM1    DS    A             Addr of User-written Handler
PARM2    DS    A             Addr of 32-bit Token
PARM3    DS    A             Addr of feedback code
*
HDLUPLST DS    0F
HDLUPRM1 DS    A             Addr of User-written Handler
HDLUPRM2 DS    A             Addr of feedback code
*
TOKEN    DS    F             32-bit Token: fullword whose
*                                 *value* will be passed
*                                 to the user handler
*                                 each time it is called.
FEEDBACK DS    CL12          CEEHDLR Feedback code
*
HDLUFBC  DS    CL12          CEEHDLU Feedback code
*
DSASIZ   EQU   *-CEEDSA      Length of DSA
         EJECT
         CEECAA ,            Map LE370 Common Anchor Area
         END   CEEHDLRA
```

CEEHDLU—Unregister user-written condition handler

CEEHDLU unregisters a user condition handler for the current stack frame. You do not necessarily need to use CEEHDLU to remove user-written condition handlers you registered with CEEHDLR. Any user-written condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by Language Environment, but only when the associated stack frame is removed from the stack.

Note: For information about restrictions on the use of CEEHDLU with PL/I, see *z/OS Language Environment Programming Guide*.

Syntax

```
►►──CEEHDLU──(──routine──,──fc──)──────────────────────────────────────►◄
```

routine **(input)**
An entry variable or constant for the routine to be unregistered as a user condition handler. This routine must be previously registered (with CEEHDLR) by the same stack frame that invokes CEEHDLU.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE07S	1	0252	CEEHDLU was unable to find the requested user-written condition handler routine.

Usage notes

- PL/I MTF consideration—CEEHDLU is not supported in PL/I MTF applications. This includes any CEEHDLR service called from a COBOL program in the application.
- z/OS UNIX consideration—In multithread applications, CEEHDLU affects only the calling thread.

For more information

- See "CEEHDLR—Register user-written condition handler" on page 317 for more information about specifying the *routine* parameter.

Examples

1. Following an example of CEEHDLU called by C/C++.

```
/*Module/File Name: EDCHDLU   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
#include <leawi.h>
#include <ceeedcct.h>

#ifdef __cplusplus
   extern "C" {
#endif
   void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
   }
#endif

int main(void) {

  _FEEDBACK fc;
  _ENTRY routine;
  _INT4 token;

  /* set the routine structure to point to the handler */
  /* and use CEEHDLR to register the user handler      */

  token = 99;
  routine.address = (_POINTER)&handler;;
  routine.nesting = NULL;

  CEEHDLR(&routine,&token,&fc);

  /* verify that CEEHDLR was successful */
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEHDLR failed with message number %d\n",
            fc.tok_msgno);
     exit (2999);
  }
 /*
⋮
  */
  /* Unregister the condition handler */
  CEEHDLU(&routine,&fc);

  /* verify that CEEHDLU was successful */
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEHDLU failed with message number %d\n",
            fc.tok_msgno);
     exit (2999);
  }
 /*
⋮
  */
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {
 /*
⋮
  */
}
```

2. Following is an example of a COBOL program that unregisters a user-written condition handler.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTHDLU
      *************************************************
      **                                             **
      ** CBLHDLU - Call CEEHDLU to unregister a user **
      **           condition handler                 **
      **                                             **
      ** In this example, a call is made to CEEHDLU  **
      ** to unregister a user condition handler      **
      ** previously registered using CEEHDLR.        **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLHDLU.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE                 PROCEDURE-POINTER.
       01  TOKEN                   PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.

       PROCEDURE DIVISION.
       PARA-CBLHDLR.
           SET ROUTINE TO ENTRY "HANDLER".
           CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY "CEEHDLR failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           ELSE
               DISPLAY "HANDLER REGISTERED"
           END-IF.

      *
 ⋮
       PARA-CBLHDLU.
           CALL "CEEHDLU" USING ROUTINE, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY "CEEHDLU failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           ELSE
               DISPLAY "HANDLER UNREGISTERED"
           END-IF.

           GOBACK.

       END PROGRAM CBLHDLU.
```

3. Following is an example of a COBOL user-written condition handler that is registered by CBLHDLR and unregistered by CBLHDLU.

```
CBL LIB,QUOTE,NOOPT,NODYNAM
      *Module/File Name: IGZTHAND
      *************************************************
      **                                             **
      ** DRVHAND - Drive sample program for COBOL    **
      **           user-written condition handler.   **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRVHAND.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE             PROCEDURE-POINTER.
```

```
 01  DENOMINATOR         PIC S9(9) BINARY.
 01  NUMERATOR           PIC S9(9) BINARY.
 01  RATIO               PIC S9(9) BINARY.
 01  TOKEN               PIC S9(9) BINARY VALUE 0.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.

 REGISTER-HANDLER.
************************************************
** Register handler                           **
************************************************
     SET ROUTINE TO ENTRY "HANDLER".
     CALL "CEEHDLR" USING ROUTINE , TOKEN , FC.
     IF  NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLR failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

 RAISE-CONDITION.
************************************************
** Cause a zero-divide condition.             **
************************************************
     MOVE 0 TO DENOMINATOR.
     MOVE 1 TO NUMERATOR.
     DIVIDE NUMERATOR BY DENOMINATOR
            GIVING RATIO.
    DISPLAY "Execution continues following "
         "divide-by-zero exception".
  UNREGISTER-HANDLER.
************************************************
** UNregister handler                         **
************************************************
     CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLU failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.
     STOP RUN.
 END PROGRAM DRVHAND.

 IDENTIFICATION DIVISION.
 PROGRAM-ID. HANDLER.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 LINKAGE SECTION.
 01  TOKEN                   PIC S9(9) BINARY.
 01  RESULT                  PIC S9(9) BINARY.
     88 RESUME                   VALUE 10.
 01  CURCOND.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
```

```
                    04  Severity    PIC S9(4) BINARY.
                    04  Msg-No      PIC S9(4) BINARY.
                03  Case-2-Condition-ID
                          REDEFINES Case-1-Condition-ID.
                    04  Class-Code  PIC S9(4) BINARY.
                    04  Cause-Code  PIC S9(4) BINARY.
                03  Case-Sev-Ctl    PIC X.
                03  Facility-ID     PIC XXX.
            02  I-S-Info            PIC S9(9) BINARY.
        01  NEWCOND.
            02  Condition-Token-Value.
            COPY  CEEIGZCT.
                03  Case-1-Condition-ID.
                    04  Severity    PIC S9(4) BINARY.
                    04  Msg-No      PIC S9(4) BINARY.
                03  Case-2-Condition-ID
                          REDEFINES Case-1-Condition-ID.
                    04  Class-Code  PIC S9(4) BINARY.
                    04  Cause-Code  PIC S9(4) BINARY.
                03  Case-Sev-Ctl    PIC X.
                03  Facility-ID     PIC XXX.
            02  I-S-Info            PIC S9(9) BINARY.
        PROCEDURE DIVISION USING CURCOND, TOKEN,
                                 RESULT, NEWCOND.

        PARA-HANDLER.
            DISPLAY "Entered user handler for condition"
               " with message number " Msg-No Of CURCOND
               " -- will resume execution".
            SET RESUME TO TRUE.

            GOBACK.
        END PROGRAM HANDLER.
```

CEEISEC—Convert integers to seconds

CEEISEC converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582. Use CEEISEC instead of CEESECS when the input is in numeric format rather than character format.

The inverse of CEEISEC is CEESECI, which converts number of seconds to integer year, month, day, hour, minute, second, and millisecond.

Syntax

```
►►—CEEISEC—(—input_year—,—input_months—,—input_day—,—input_hours————►

►—,—input_minutes—,—input_seconds—,—input_milliseconds—,——————————————►

►—output_seconds—,—fc—)——————————————————————————————————————————————►◄
```

input_year **(input)**
A 32-bit binary integer representing the year. The valid range for *input_year* is 1582 to 9999, inclusive.

`input_month` **(input)**
A 32-bit binary integer representing the month. The valid range for *input_month* is 1 to 12.

`input_day` **(input)**
A 32-bit binary integer representing the day. The valid range for *input_day* is 1 to 31.

`input_hours` **(input)**
A 32-bit binary integer representing the hours. The range of valid *input_hours* is 0 to 23.

`input_minutes` **(input)**
A 32-bit binary integer representing the minutes. The range of valid *input_minutes* is 0 to 59.

`input_seconds` **(input)**
A 32-bit binary integer representing the seconds. The range of valid *input_seconds* is 0 to 59.

`input_milliseconds` **(input)**
A 32-bit binary integer representing milliseconds. The range of valid *input_milliseconds* is 0 to 999.

`output_seconds` **(output)**
A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range of *output_seconds* is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999). If any input values are invalid, *output_seconds* is set to zero. To convert *output_seconds* to a Lilian day number, divide *output_seconds* by 86,400 (the number of seconds in a day).

`fc` **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2EE	3	2510	The hours value in a call to CEEISEC or CEESECS was not recognized.
CEE2EF	3	2511	The day parameter passed in a CEEISEC call was invalid for year and month specified.
CEE2EH	3	2513	The input date passed in a CEEISEC, CEEDAYS, or CEESECS call was not within the supported range.
CEE2EI	3	2514	The year value passed in a CEEISEC call was not within the supported range.
CEE2EJ	3	2515	The milliseconds value in a CEEISEC call was not recognized.
CEE2EK	3	2516	The minutes value in a CEEISEC call was not recognized.
CEE2EL	3	2517	The month value in a CEEISEC call was not recognized.
CEE2EN	3	2519	The seconds value in a CEEISEC call was not recognized.

Usage notes

- z/OS UNIX consideration—In multithread applications, CEEISEC affects only the calling thread.

For more information

- For more information about the CEESECS callable service, see "CEESECS—Convert timestamp to seconds" on page 405.
- See "CEESECI—Convert seconds to integers" on page 400 for more information about the CEESECI callable service.

Examples

1. Following is an example of CEEISEC called by C/C++.

```
/*Module/File Name: EDCISEC   */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

  _INT4 year, month, day, hours, minutes, seconds,
        millisecs;
  _FLOAT8 output;
  _FEEDBACK fc;

  year = 1991;
  month = 9;
  day = 13;
  hours = 4;
  minutes = 34;
  seconds = 25;
  millisecs = 746;

  CEEISEC(&year,&month,&day,&hours,&minutes,&seconds,;
          &millisecs,&output,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEISEC failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  printf("The number of seconds between 00:00:00.00"
        " 10/14/1582 and 04:34:25.746 09/13/1991"
        " is %.3f\n",output);
}
```

2. Following is an example of CEEISEC called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTISEC
      *************************************************
      **                                             **
      ** CBLISEC - Call CEEISEC to convert integers  **
      **           to seconds                        **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLISEC.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  YEAR                   PIC S9(9) BINARY.
       01  MONTH                  PIC S9(9) BINARY.
```

```
 01  DAYS                    PIC S9(9) BINARY.
 01  HOURS                   PIC S9(9) BINARY.
 01  MINUTES                 PIC S9(9) BINARY.
 01  SECONDS                 PIC S9(9) BINARY.
 01  MILLSEC                 PIC S9(9) BINARY.
 01  OUTSECS                 COMP-2.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION.
 PARA-CBLISEC.
*************************************************
** Specify seven binary integers representing  **
** the date and time as input to be converted  **
** to Lilian seconds                           **
*************************************************
     MOVE 2000 TO YEAR.
     MOVE 1 TO MONTH.
     MOVE 1 TO DAYS.
     MOVE 0 TO HOURS.
     MOVE 0 TO MINUTES.
     MOVE 0 TO SECONDS.
     MOVE 0 TO MILLSEC.
*************************************************
** Call CEEISEC to convert the integers        **
** to seconds                                  **
*************************************************
     CALL "CEEISEC" USING YEAR, MONTH, DAYS,
                          HOURS, MINUTES, SECONDS,
                          MILLSEC, OUTSECS , FC.

*************************************************
** If CEEISEC runs successfully, display result**
*************************************************
     IF CEE000 of FC  THEN
         DISPLAY MONTH "/" DAYS "/" YEAR
             " AT " HOURS ":" MINUTES ":" SECONDS
             " is equivalent to " OUTSECS " seconds"
     ELSE
         DISPLAY "CEEISEC failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

3. Following is an example of CEEISEC called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMISEC
 /****************************************************/
 /**                                                **/
 /** Function: CEEISEC - Convert integers to        **/
 /**                     seconds                    **/
 /**                                                **/
 /** In this example, CEEISEC is called to convert  **/
 /** integers representing the date and time to the **/
 /** number of seconds since 00:00 14 October 1582. **/
/**                                                 **/
/*****************************************************/
PLIISEC: PROC OPTIONS(MAIN);
   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL YEAR     REAL FIXED BINARY(31,0);
   DCL MONTH    REAL FIXED BINARY(31,0);
   DCL DAYS     REAL FIXED BINARY(31,0);
   DCL HOURS    REAL FIXED BINARY(31,0);
   DCL MINUTES  REAL FIXED BINARY(31,0);
   DCL SECONDS  REAL FIXED BINARY(31,0);
   DCL MILLSEC  REAL FIXED BINARY(31,0);
   DCL OUTSECS  REAL FLOAT DECIMAL(16);
   DCL 01 FC,                     /* Feedback token */
          03 MsgSev    REAL FIXED BINARY(15,0),
          03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),      /* Facility ID */
          03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);
   /* Specify integers representing                 */
   /*    00:00:00 1 January 2000                    */

   YEAR = 2000;
   MONTH = 1;
   DAYS = 1;
   HOURS = 0;
   MINUTES = 0;
   SECONDS = 0;
   MILLSEC = 0;
  /* Call CEEISEC to convert integers to Lilian     */
  /* seconds                                        */

   CALL CEEISEC ( YEAR, MONTH, DAYS, HOURS,
      MINUTES, SECONDS, MILLSEC, OUTSECS, FC );
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT EDIT( OUTSECS, ' seconds corresponds to ',
          MONTH, '/', DAYS, '/', YEAR, ' at ', HOURS,
          ':', MINUTES, ':', SECONDS, '.', MILLSEC )
          (SKIP, F(9), A, 2 (P'99',A), P'9999', A,
          3 (P'99', A), P'999' );
      END;
   ELSE  DO;
      DISPLAY( 'CEEISEC failed with msg '
         || FC.MsgNo );
      STOP;
      END;

END PLIISEC;
```

CEEITOK—Return initial condition token

CEEITOK returns the condition that initially triggered the current condition. The current condition might be different from the initial condition if the initial condition has been promoted by a user-written condition handler.

Syntax

```
►►──CEEITOK──(──i_ctok──,──fc──)────────────────────────────────────────►◄
```

i_ctok **(output)**
A 12-byte condition token identifying the initial condition in the current active data block being processed.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE35S	1	3260	No condition was active when a call to a condition management routine was made.

Usage notes

- z/OS UNIX considerations—In multithread applications, CEEITOK affects only the calling thread. CEEITOK returns the initial token for the condition of the thread.

For more information

- See "CEENCOD—Construct a condition token" on page 377, for more information about the CEENCOD callable service.

Examples

1. Following is an example of CEEITOKC called by C/C++.

```
/*Module/File Name: EDCITOK   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

#ifdef __cplusplus
   extern "C" {
#endif
   void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
   }
#endif

int main(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;
  /* register condition handler */
  token = 99;
  routine.address = (_POINTER)&handler;;
  routine.nesting = NULL;
  CEEHDLR(&routine,&token,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEEHDLR failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
  /* .
     .
     . */
  /* build the condition token */
  c_1 = 1;
  c_2 = 99;
  cond_case = 1;
  sev = 1;
  control = 0;
  memcpy(facid,"ZZZ",3);
  isi = 0;
  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&condtok,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  /*
⋮
 */
  /* signal the condition */
  CEESGL(&condtok,&qdata,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEESGL failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
  /*
⋮
 */
}
void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _FEEDBACK orig_fc, itok_fc;
  /*
⋮
 */
   /* get the original condition token */
   CEEITOK(&orig_fc, &itok_fc);
  if ( _FBCHECK ( itok_fc , CEE000 ) != 0 ) {
      printf("CEEITOK failed with message number %d\n",
             itok_fc.tok_msgno);
      exit(2999);
   }
  /*
⋮
 */
  *result = 10;
}
```

2. Following is an example of CEEITOKC called by COBOL.

```
CBL LIB,QUOTE,NOOPT
      *Module/File Name: IGZTITOK
      *************************************************
      **                                             **
      ** Purpose:  Drive sample program for CEEITOK. **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRVITOK.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE          PROCEDURE-POINTER.
       01  DENOMINATOR      PIC S9(9) BINARY.
       01  NUMERATOR        PIC S9(9) BINARY.
       01  RATIO            PIC S9(9) BINARY.
       01  TOKEN            PIC S9(9) BINARY VALUE 0.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.

       PROCEDURE DIVISION.

       REGISTER-HANDLER.
      *************************************************
      ** Register handler                            **
      *************************************************
           SET ROUTINE TO ENTRY "CBLITOK".
           CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
           IF  NOT CEE000 of FC  THEN
               DISPLAY "CEEHDLR failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

       RAISE-CONDITION.
      *************************************************
      ** Cause a zero-divide condition.              **
      *************************************************
           MOVE 0 TO DENOMINATOR.
           MOVE 1 TO NUMERATOR.
           DIVIDE NUMERATOR BY DENOMINATOR, GIVING RATIO.
       UNREGISTER-HANDLER.
      *************************************************
      ** UNregister handler                          **
      *************************************************
           CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY "CEEHDLU failed with msg "
                   Msg-No of FC UPON CONSOLE
           END-IF.
           STOP RUN.

       END PROGRAM DRVITOK.

      *************************************************
      **                                             **
      ** Function: CEEITOK - Return initial          **
**                    condition token         **
**                                            **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLITOK.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ITOKEN.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 LINKAGE SECTION.
 01  TOKEN                   PIC S9(9) BINARY.
 01  RESULT                  PIC S9(9) BINARY.
     88 RESUME                   VALUE 10.
 01  CURCOND.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 01  NEWCOND.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING CURCOND, TOKEN,
                          RESULT, NEWCOND.
 PARA-CBLITOK.
```

```
         CALL "CEEITOK" USING ITOKEN, FC.
         IF  CEE000 of FC  THEN
             DISPLAY "Initial condition has msg "
                 Msg-No of ITOKEN
         ELSE
             DISPLAY "CEEITOK failed with msg "
                 Msg-No of FC UPON CONSOLE
             STOP RUN
         END-IF.
          PARA-HANDLER.
    ************************************************
    ** In user handler - resume execution         **
    ************************************************
         SET RESUME TO TRUE.

         GOBACK.

     END PROGRAM CBLITOK.
```

3. Following is an example of CEEITOKC called by PL/I.

```
*PROCESS OPT(0), MACRO;
 /* Module/File Name: IBMITOK                          */
 /****************************************************/
 /**                                                **/
 /** Function: CEEITOK - example of CEEITOK         **/
 /**                    invoked from PL/I ON-unit  **/
 /**                                                **/
 /****************************************************/
 IBMITOK:  PROCEDURE  OPTIONS(MAIN);
   %INCLUDE CEEIBMAW;
   %INCLUDE CEEIBMCT;
   DECLARE
      01 ITOKEN,                   /* Feedback token */
         03 MsgSev    REAL FIXED BINARY(15,0),
         03 MsgNo     REAL FIXED BINARY(15,0),
         03 Flags,
            05 Case      BIT(2),
            05 Severity  BIT(3),
            05 Control   BIT(3),
         03 FacID     CHAR(3),        /* Facility ID */
         03 ISI     /* Instance-Specific Information */
                  REAL FIXED BINARY(31,0),
      01 FC,                       /* Feedback token */
         03 MsgSev    REAL FIXED BINARY(15,0),
         03 MsgNo     REAL FIXED BINARY(15,0),
         03 Flags,
            05 Case      BIT(2),
            05 Severity  BIT(3),
            05 Control   BIT(3),
         03 FacID     CHAR(3),        /* Facility ID */
         03 ISI     /* Instance-Specific Information */
                  REAL FIXED BINARY(31,0),
      divisor     FIXED BINARY(31) INITIAL(0);
   ON  ZERODIVIDE  BEGIN;
      CALL CEEITOK ( ITOKEN, FC );
      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST('The initial condition for the '
            || 'current active block was message '
            || ITOKEN.MsgNo
            || ' for facility '|| ITOKEN.FacID );
         END;
      ELSE  DO;
         DISPLAY( 'CEEITOK failed with msg '
            || FC.MsgNo );
         CALL  CEEMSG( FC, 2, * );
```

```
            END;
         END /* ON ZeroDivide */;
      divisor = 15 / divisor  /* signals ZERODIVIDE */;
    END IBMITOK;
```

CEELCNV—Query locale numeric conventions

CEELCNV, which calls the C function `localeconv()`, queries the numeric formatting information from the current locale and sets the components of a structure with values pertaining to the LC_NUMERIC and LC_MONETARY categories. It sets the components of an object of the type NM_STRUCT with the values appropriate for the formatting of the numeric quantities (monetary and otherwise) according to the rules of the current locale.

CEELCNV is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►—CEELCNV—(—version—,—num_and_mon—,—fc—)——————————————————————►◄
```

version
This parameter points to a user supplied VERSION_INFO structure, in which the first four bytes contain the callable service version number.

Requirement: If the parameter does not point to a user supplied VERSION_INFO structure, it must be omitted.If the parameter points to a VERSION_INFO containing version number 2, CEELCNV returns a num_and_mon structure with the international monetary string formatting elements added for ISO/IEC 9899:1999 (C99). This corresponds to the current C language `lconv` structure. If the parameter is omitted, CEELCNV returns the structure format used before C99. For information about how to code this parameter, see "Invoking callable services" on page 114.

num_and_mon **(output)**
Points to the numeric and monetary structure that is filled in by this service. If the service fails, the contents of the structure are undefined. *num_and_mon* has the following structure:

NM_STRUCT
A halfword length-prefixed character string (VSTRING). A pointer to the filled-in structure NM_STRUCT is returned. The structure pointed to by the return value should not be modified by the program but can be overridden by subsequent calls to CEELCNV. In addition, calls to CEESETL with the LC_ALL, LC_MONETARY or LC_NUMERIC categories can cause subsequent calls to CEELCNV to return different values based on the selection of the locale.

The members of the structure with the type VSTRING are strings, any of which (except decimal_point) can point to an empty string, to indicate that the value is not available in the current locale or is of zero length. The members with type VINT are non-negative numbers, any of which can be CHAR_MAX to indicate that the value is not available in the current locale. The members include the following:

VSTRING decimal_point
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the decimal-point character used to format non-monetary quantities.

VSTRING thousands_sep
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the character used to separate groups of digits to the left of the decimal point in formatted non-monetary quantities.

VSTRING grouping
A halfword length-prefixed character string (VSTRING) of 22 bytes whose elements indicate the size of each group of digits in formatted non-monetary quantities.

VSTRING int_curr_symbol
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the international currency symbol applicable to the current locale, left justified within a four-character space-padded field.

VSTRING currency_symbol
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the local currency symbol applicable to the current locale.

VSTRING mon_decimal_point
A halfword length-prefixed character string(VSTRING) of 22 bytes that is the decimal point character used to format monetary quantities.

VSTRING mon_thousands_sep
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the separator for groups of digits to the left of the decimal point in formatted monetary quantities.

VSTRING mon_grouping
A halfword length-prefixed character string (VSTRING) of 22 bytes whose elements indicate the size of each group of digits in formatted monetary quantities.

VSTRING positive_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes that indicates a non-negative-formatted monetary quantity.

VSTRING negative_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes used to indicate a negative-formatted monetary quantity.

VINT int_frac_digits
A 1-byte integer that is the number of fractional digits (those to the right of the decimal point) to be displayed in an internationally-formatted monetary quantity.

VINT frac_digits
A 1-byte integer that is the number of fractional digits (those to the right of the decimal point) to be displayed in a formatted monetary quantity.

VINT p_cs_precedes
A 1-byte integer that is set to 1 if the *currency_symbol* precedes the value for a non-negative-formatted monetary quantity. It is set to 0 if it follows.

`VINT p_sep_by_space`
A 1-byte integer that is set to 1 if the *currency_symbol* is separated by a space from the value for a non-negative-formatted monetary quantity. It is set to 0 if it is not separated.

`VINT n_cs_precedes`
A 1-byte integer that is set to 1 if the *currency_symbol* precedes the value for a negative-formatted monetary quantity. It is set to 0 if it follows.

`VINT n_sep_by_space`
A 1-byte integer that is set to 1 if the *currency_symbol* is separated by a space from the value for a negative-formatted monetary quantity. It is set to 0 is it is not separated.

`VINT p_sign_posn`
A 1-byte integer that is set to a value indicating the positioning of the *positive_sign* for non-negative-formatted monetary quantity.

`VINT n_sign_posn`
A 1-byte integer that is set to a value indicating the position of the *negative_sign* for negative-formatted monetary quantity.

`VSTRING left_parenthesis`
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the used to denote a negative monetary quantity.

`VSTRING right_parenthesis`
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the used to denote a negative monetary quantity.

`VSTRING debit_sign`
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the debit sign character in monetary formats.

`VSTRING credit_sign`
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the credit sign character in monetary formats.

`VINT int_p_cs_precedes`
A 1-byte integer that is set to 1 if the *currency_symbol* precedes the value for a non-negative-formatted international monetary quantity. It is set to 0 if it follows.

`VINT int_p_sep_by_space`
A 1-byte integer that is set to 1 if the *currency_symbol* is separated by a space from the value for a non-negative-formatted international monetary quantity. It is set to 0 if it is not separated.

`VINT int_n_cs_precedes`
A 1-byte integer that is set to 1 if the *currency_symbol* precedes the value for a negative-formatted international monetary quantity. It is set to 0 if it follows.

`VINT int_n_sep_by_space`
A 1-byte integer that is set to 1 if the *currency_symbol* is separated by a space from the value for a negative-formatted international monetary quantity. It is set to 0 is it is not separated.

VINT int_p_sign_posn
: A 1-byte integer that is set to a value indicating the positioning of the *positive_sign* for non-negative-formatted international monetary quantity.

VINT n_sign_posn
: A 1-byte integer that is set to a value indicating the position of the *negative_sign* for negative-formatted international monetary quantity.

fc **(output/optional)**
: A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.
CEE3VN	3	4087	Input Error: The `version_info` control block contains a version number that is not valid.

Usage notes

- PL/I MTF consideration—CEELCNV is not supported in PL/I MTF applications.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.
- CEELCNV does not return all numeric conventions.
- CHAR_MAX determines when no further grouping is to be performed. The elements of *grouping* and *mon_grouping* are interpreted according to the following CHAR_MAX settings:
  - 0 specifies that the previous element is to be repeatedly used for the remainder of the digits. Indicates that no further grouping is to be performed.
  - Any other value represents the number of digits comprising the current group. The next element is examined to determine the size of the next group of digits to the left of the current group.
- The value of *p_sign_posn* and *n_sign_posn* is interpreted according to the following:

  0 Parentheses surround the quantity and *currency_symbol*.

  1 The sign string precedes the quantity and *currency_symbol*.

  2 The sign string follows the quantity and *currency_symbol*.

  3 The sign string immediately precedes the *currency_symbol*.

  4 The sign string immediately follows the *currency_symbol*.

For more information

- See "CEEQRYL—Query active locale environment" on page 389 for a description of LC_NUMERIC and LC_MONETARY catagories.
- See "CEESETL—Set locale operating environment" on page 411 for details on the CEESETL callable service.

Examples

1. Following is an example of CEELCNV called by COBOL.

```
  CBL LIB,QUOTE
    *Module/File Name: IGZTLCNV
    *************************************************
    ** Example for callable service CEELCNV        **
    **  Function: Retrieve numeric and monetary    **
    **             format for default locale and   **
    **             print an item.                  **
    **            Set locale to France, retrieve   **
    **             structure, and print an item.   **
    ** Valid only for COBOL for MVS & VM Release 2 **
    ** or later.                                   **
    *************************************************
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  MAINLCNV.
     DATA DIVISION.
     WORKING-STORAGE SECTION.
    *************************************************
    ** Use Locale NM-Struct for CEELCNV calls      **
    *************************************************
     COPY CEEIGZN2M.
    *
     PROCEDURE DIVISION.
    *************************************************
    **  Subroutine needed for addressing           **
    *************************************************
         CALL "COBLCNV" USING NM-Struct.
         STOP RUN.
    *
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  COBLCNV.
     DATA DIVISION.
     WORKING-STORAGE SECTION.
     01  PTR1 Pointer.
     01  Locale-Name.
         02  LN-Length  PIC S9(54) BINARY.
         02  LN-String  PIC X(256).

    *************************************************
    ** Use Locale category constants              **
    *************************************************
     COPY CEEIGZLC.
    *
     01  FC.
         02  Condition-Token-Value.
         COPY CEEIGZCT.
             03  Case-1-Condition-ID.
                 04  Severity    PIC S9(4) BINARY.
                 04  Msg-No      PIC S9(4) BINARY.
             03  Case-2-Condition-ID
              REDEFINES Case-1-Condition-ID.
                 04  Class-Code  PIC S9(4) BINARY.
                 04  Cause-Code  PIC S9(4) BINARY.
             03  Case-Sev-Ctl    PIC X.
             03  Facility-ID     PIC XXX.
         02  I-S-Info            PIC S9(9) BINARY.
LINKAGE SECTION.
    *************************************************
    ** Use Locale NM-Struct for CEELCNV calls      **
    *************************************************
     COPY CEEIGZN2M.
    *
     PROCEDURE DIVISION USING NM-Struct.
    *************************************************
    **  Call CEELCNV to retrieve values for locale**
    *************************************************
         CALL "CEELCNV" USING OMITTED,
                        ADDRESS OF NM-Struct, FC.
```

```
*************************************************
**  Check feedback code and display result     **
*************************************************
     IF Severity = 0 THEN
        DISPLAY "Default decimal point is "
        DECIMAL-PT-String(1:DECIMAL-PT-Length)
     ELSE
        DISPLAY "Call to CEELCNV failed. " Msg-No
     END-IF.
*************************************************
** Set up locale for France                    **
*************************************************
     MOVE 54 TO LN-Length.
     MOVE "Fr_FR" TO LN-String (1:LN-Length).
*************************************************
**  Call CEESETL to set monetary locale        **
*************************************************
     CALL "CEESETL" USING Locale-Name,
           LC-MONETARY, FC.

*************************************************
**  Call CEESETL to set numeric locale         **
*************************************************
     CALL "CEESETL" USING Locale-Name,
           LC-NUMERIC, FC.
*************************************************
**  Check feedback code and call CEELCNV again **
**  using version 2 to get at C99 mapping.     **
*************************************************
     IF Severity = 0
      MOVE 2 TO Version
        set PTR1 to address of Version-Info
        CALL "CEELCNV" USING PTR1,
                       ADDRESS OF NM-Struct, FC
        IF Severity > 0
           DISPLAY "Call to CEELCNV failed. "
                   Msg-No
        ELSE
           DISPLAY "French decimal point is "
           DECIMAL-PT-String(1:DECIMAL-PT-Length)
        END-IF
     ELSE
        DISPLAY "Call to CEESETL failed. " Msg-No
     END-IF.
     EXIT PROGRAM.
 END PROGRAM COBLCNV.
*
 END PROGRAM MAINLCNV.
```

2. Following is an example of CEELCNV called by PL/I.

```
*PROCESS MACRO;
     /*Module/File Name: IBMLCNV                        */
     /***************************************************/
     /* Example for callable service CEELCNV            */
     /* Function: Retrieve numeric and monetary format  */
     /*  structure for default locale and print an item. */
     /*  Set locale to France, retrieve structure and   */
     /*  print an item.                                 */
     /***************************************************/
     PLILCNV: PROC OPTIONS(MAIN);
     %INCLUDE CEEIBMAW; /* ENTRY defs, macro defs       */
     %INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants   */
     %INCLUDE CEEIBMLC; /* Locale category constants     */
     %INCLUDE CEEIBMN2M; /* NM_STRUCT for CEELCNV calls   */
     /* use explicit pointer for local NM_STRUCT struct  */
     DCL NUM_AND_MON POINTER INIT(ADDR(NM_STRUCT));
```

```
/* Point to local version_info struct and initialize*/
/* VERSION TO 2 TO USE C99 MAPPING OF NM_STRUCT     */
DCL VERSN POINTER INIT(ADDR(version_info));
VERSION_INFO.VERSION = 2;

/* CEESETL service call arguments */
DCL LOCALE_NAME CHAR(256) VARYING;

DCL 01 FC,                         /* Feedback token */
       03 MsgSev    REAL FIXED BINARY(15,0),
       03 MsgNo     REAL FIXED BINARY(15,0),
       03 Flags,
          05 Case      BIT(2),
          05 Severity  BIT(3),
          05 Control   BIT(3),
       03 FacID     CHAR(3),       /* Facility ID */
       03 ISI       /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);
  /* retrieve structure for default locale */
  CALL CEELCNV ( *, NUM_AND_MON, FC );
  PUT SKIP LIST('Default DECIMAL_POINT is ',
               NM_STRUCT.DECIMAL_POINT);
  /* set locale for France */
    LOCALE_NAME = 'Fr_FRFRAN';
  /* use LC_NUMERIC category const from CEEIBMLC */
  CALL CEESETL ( LOCALE_NAME, LC_NUMERIC,  FC );
  /* use LC_MONETARY category const from CEEIBMLC */
  CALL CEESETL ( LOCALE_NAME, LC_MONETARY,  FC );
  /* FBCHECK macro used (defined in CEEIBMCT) */
  IF FBCHECK( FC, CEE000 ) THEN
    DO;
      /* retrieve active NM_STRUCT, France Locale */
      CALL CEELCNV ( *VERSN, NUM_AND_MON, FC );
      PUT SKIP LIST('French DECIMAL_POINT is ',
                   NM_STRUCT.DECIMAL_POINT);
    END;
END PLILCNV;
```

CEELOCT—Get current local date or time

CEELOCT returns the current local date or time in three formats:

- Lilian date (the number of days since 14 October 1582)
- Lilian seconds (the number of seconds since 00:00:00 14 October 1582)
- Gregorian character string (in the form `YYYYMMDDHHMISS999`)

These values are compatible with other Language Environment date and time services, and with existing language intrinsic functions.

CEELOCT performs the same function as calling the CEEGMT, CEEGMTO, and CEEDATM date and time services separately. CEELOCT, however, performs the same services with much greater speed.

The character value returned by CEELOCT is designed to match that produced by existing language intrinsic functions. The numeric values returned can be used to simplify date calculations.

Syntax

```
►►──CEELOCT──(──output_Lilian──,──output_seconds──,──output_Gregorian──,──────►

►─fc──)─────────────────────────────────────────────────────────────────────►◄
```

output_Lilian **(output)**
A 32-bit binary integer representing the current local date in the Lilian format, that is, day 1 equals 15 October 1582, day 148,887 equals 4 June 1990. If the local time is not available from the system, *output_Lilian* is set to 0 and CEELOCT terminates with a non-CEE000 symbolic feedback code.

output_seconds **(output)**
A 64-bit double-floating point number representing the current local date and time as the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). 19:00:01.078 on 4 June 1990 is second number 12,863,905,201.078. If the local time is not available from the system, *output_seconds* is set to 0 and CEELOCT terminates with a non-CEE000 symbolic feedback code.

output_Gregorian **(output)**
A 17-byte fixed-length character string in the form YYYYMMDDHHMISS999 representing local year, month, day, hour, minute, second, and millisecond. If the format of *output_Gregorian* does not meet your needs, you can use the CEEDATM callable service to convert *output_seconds* to another format.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2F3	3	2531	The local time was not available from the system.

Usage notes

- z/OS consideration—The MVS command SET DATE will not affect the value returned by CEELOCT.
- CICS consideration—CEELOCT does not use the OS TIME macro.
- z/OS UNIX consideration—In multithread applications, CEELOCT affects only the calling thread.

For more information

- See "CEEDATM—Convert seconds to character timestamp" on page 236 for more information about the CEEDATM callable service.

Examples

1. Following is an example of CEELOCT called by C/C++.

```
/*Module/File Name: EDCLOCT   */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _INT4     lil_date;
  _FLOAT8   local_date;
  _CHAR17   gregorian_date;

  CEELOCT(&lil_date,&local_date,gregorian_date,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEELOCT failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  printf("The current date is YYYYMMDDHHMISS999\n");
  printf("                    %.17s\n",gregorian_date);
}
```

2. Following is an example of CEELOCT called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTLOCT
      ************************************************
      **                                            **
      ** CBLLOCT - Call CEELOCT to get current      **
      **           local time                       **
      **                                            **
      ** In this example, a call is made to CEELOCT **
      ** to return the current local time in Lilian **
      ** days (the number of days since 14 October  **
      ** 1582), Lilian seconds (the number of       **
      ** seconds since 00:00:00 14 October 1582),   **
      ** and a Gregorian string (in the form        **
      ** YYYMMDDMISS999). The Gregorian character   **
      ** string is then displayed.                  **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLLOCT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  LILIAN                 PIC S9(9) BINARY.
       01  SECONDS                COMP-2.
       01  GREGORN                PIC X(17).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info           PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLLOCT.
           CALL "CEELOCT" USING LILIAN, SECONDS,
                              GREGORN, FC.
      ************************************************
      ** If CEELOCT runs successfully, display      **
      **     Gregorian character string             **
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY "Local Time is " GREGORN
           ELSE
               DISPLAY "CEELOCT failed with msg "
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

3. Following is an example of CEELOCT called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMLOCT                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEELOCT - get current local time     **/
 /**                                                **/
 /** In this example, CEELOCT is called to return   **/
 /** the current local time as a Lilian date,       **/
 /** Lilian timestamp, and Gregorian character      **/
 /** string.                                        **/
 /**                                                **/
 /****************************************************/
 PLILOCT: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL LILIAN   REAL FIXED BINARY(31,0);
    DCL SECONDS  REAL FLOAT DECIMAL(16);
    DCL GREGORN  CHARACTER ( 17 );
    DCL 01 FC,                        /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);

   /* Call CEELOCT to return local time in 3 formats */
    CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC );

    /* If CEELOCT ran successfully, print Gregorian  */
    /* result                                        */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'The local date and time are '
          || GREGORN || '.' );
       END;
    ELSE  DO;
       DISPLAY( 'CEELOCT failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLILOCT;
```

CEEMGET—Get a message

CEEMGET retrieves, formats, and stores in a passed message area a message corresponding to a condition token that is either returned from a callable service or passed to a user-written condition handler. The caller can later retrieve the message to change or to write as output.

Syntax

```
►►──CEEMGET──(──cond_token──,──message_area──,──msg_ptr──,──fc──)──────────►◄
```

cond_token **(input/output)**
A 12-byte condition token received or returned as the result of a Language Environment callable service.

message_area **(input/output)**
A fixed-length 80-character string (VSTRING), where the message is placed. The message is left-justified and padded on the right with blanks.

msg_ptr **(input/output)**
A 4-byte binary integer returned to the calling routine. The *msg_ptr* should be passed a value of zero on the initial call to CEEMGET. If a message is too large to be contained in the *message_area, msg_ptr* (containing the index) is returned into the message. This index is used on subsequent calls to CEEMGET to retrieve the remaining portion of the message. A feedback code is also returned, indicating that the message was truncated. When the entire message is returned, *msg_ptr* is zero.

The *msg_ptr* contains different results based on the length of the message:

- If a message contains fewer than 80 characters, the entire message is returned on the first call. *msg_ptr* contains 0.
- If a message contains exactly 80 characters, the entire message is returned on the first call. *msg_ptr* contains 0.
- If the message is too long, CEEMGET splits it into segments. The *msg_ptr* does not contain the cumulative index for the entire message returned so far, but contains only the index into the segment that was just returned. It is up to the user of CEEMGET to maintain the cumulative count if needed. When a message is too long, the following can occur:
  - If a message contains more than 80 characters and at least one blank is contained in the first 80 characters, the string up to and including the last blank is returned on the first call.
  - If the 80th character is nonblank (even if the 81st character is a blank), *msg_ptr* contains the index of the last blank (something less than 80), and the next call starts with the next character.
  - If the 80th character is a blank, *msg_ptr* contains 80 and the next call starts with the 81st character, blank or nonblank.
  - If a message contains more than 80 characters and at least the first 80 are all nonblank, the first 80 are returned and the next call does not add any blanks and starts with the 81st character. *msg_ptr* contains 80.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result

of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE036	3	0102	An unrecognized condition token was passed to *routine* and could not be used.
CEE0E2	3	0450	The message inserts for the condition token with message number *message-number* and facility ID *facility-id* could not be located.
CEE0E6	3	0454	The message number *message-number* could not be found for facility ID *facility-id*.
CEE0E7	1	0455	The message with message number *message-number* and facility ID *facility-id* was truncated.
CEE0EA	3	0458	The message repository *repository-name* could not be located.

Usage notes

- z/OS UNIX considerations—In multithread applications, CEEMGET affects only the calling thread. However, CEEMGET uses the NATLANG value of the enclave. Any subsequent calls to CEEMGET, for a given condition, use the NATLANG value in effect at the time of the first call.

For more information

- See "CEENCOD—Construct a condition token" on page 377 for a description of the 12-byte condition token constructed by the CEENCOD callable service.

Examples

1. Following is an example of CEEMGET called by C/C++.

```
/*Module/File Name: EDCMGET   */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

  _VSTRING message;
  _INT4 dest,msgindx;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;
  _CHAR80 msgarea;
  _FEEDBACK fc,token;

  /* construct a token for CEE message 2523 */
  c_1 = 1;
  c_2 = 2523;
  cond_case = 1;
  sev = 1;
  control = 1;
  memcpy(facid,"CEE",3);
  isi = 0;
  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&token,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  msgindx = 0;
  memset(msgarea,' ',79);/* initialize the message area */
  msgarea_80¨ = '\0';

  /* use CEEMGET until all the message has been */
  /* retrieved */
  /* msgindx will be zero when all the message has */
  /* been retrieved */
  do {
     CEEMGET(&token,msgarea,&msgindx,&fc);

     if (fc.tok_sev > 1 ) {
        printf("CEEMGET failed with message number %d\n",
               fc.tok_msgno);
        exit(2999);
     }
     /* put out the message using CEEMOUT */
     memcpy(message.string,msgarea,80);
     message.length = 80;
     dest = 2;
     CEEMOUT(&message,&dest,&fc);

     if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEMOUT failed with message number %d\n",
               fc.tok_msgno);
        exit(2999);
     }
  } while (msgindx != 0);
}
```

2. Following is an example of CEEMGET called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTMGET
      ******************************************
      **                                      **
      ** CBLMGET - Call CEEMGET to get a      **
      **           message.  First set up a   **
      **           condition token using      **
      **           CEENCOD.                   **
      **                                      **
      ******************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLMGET.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  MSGBUF                  PIC X(80).
       01  MSGPTR                  PIC S9(9) BINARY.
       01  SEV                     PIC S9(4) BINARY.
       01  MSGNO                   PIC S9(4) BINARY.
       01  CASE                    PIC S9(4) BINARY.
       01  SEV2                    PIC S9(4) BINARY.
       01  CNTRL                   PIC S9(4) BINARY.
       01  FACID                   PIC X(3).
       01  ISINFO                  PIC S9(9) BINARY.
       01  NEWTOK.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
```

```
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    01  FC.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    PROCEDURE DIVISION.
    PARA-CBLMGET.

   ********************************************
   ** Give contok value of                  **
   ** sev = 0, msgno = 1 facid = CEE        **
   ********************************************
        MOVE 0 TO SEV.
        MOVE 1 TO MSGNO.
        MOVE 1 TO CASE.
        MOVE 0 TO SEV2.
        MOVE 1 TO CNTRL.
        MOVE "CEE" TO FACID.
        MOVE 0 TO ISINFO.

   *************************************************
   ** Call CEENCOD with the values assigned above **
   ** to build a condition token "NEWTOK"         **
   *************************************************
        CALL "CEENCOD" USING SEV, MSGNO, CASE,
                             SEV2, CNTRL, FACID,
                             ISINFO, NEWTOK, FC.
        IF NOT CEE000 of FC  THEN
            DISPLAY "CEENCOD failed with msg "
                Msg-No of FC UPON CONSOLE
            STOP RUN
        END-IF.

   *************************************************
   ** Always pass 0 in MSGPTR on the initial      **
   ** call to CEEMGET.  If the message is too     **
   ** long to be returned in a single call,       **
   ** MSGPTR will be returned containing an       **
   ** index to the message that can be used on    **
   ** subsequent calls to CEEMGET.                **
   *************************************************
        MOVE 0 TO MSGPTR.
   *************************************************
   ** Call CEEMGET to get the message associated  **
   ** with the condition token                    **
   *************************************************
        CALL "CEEMGET" USING NEWTOK, MSGBUF,
                             MSGPTR , FC.
        IF NOT CEE000 of FC  THEN
            DISPLAY "CEEMGET failed with msg "
```

```
                  Msg-No of FC UPON CONSOLE
              STOP RUN
          ELSE
              DISPLAY "The message is: " MSGBUF
          END-IF.

          GOBACK.
```

3. Following is an example of CEEMGET called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMMGET                         */
 /****************************************************/
 /**                                                **/
 /**Function       : CEEMGET - Get a Message         **/
 /**                                                **/
 /****************************************************/
 PLIMGET: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL 01 CONTOK,                 /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    DCL MSGBUF      CHAR(80);
    DCL MSGPTR      REAL FIXED BINARY(31,0);

    /* Give CONTOK value of condition CEE001         */
    ADDR( CONTOK ) -> CEEIBMCT = CEE001;
    MSGPTR = 0;

    /* Call CEEMGET to retrieve msg corresponding    */
    /*    to condition token                         */
    CALL CEEMGET ( CONTOK, MSGBUF, MSGPTR, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'Message text for message number'
          || CONTOK.MsgNo || ' is "' || MSGBUF || '"');
       END;
    ELSE  DO;
       DISPLAY( 'CEEMGET failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLIMGET;
```

CEEMOUT—Dispatch a message

CEEMOUT dispatches a user-defined message string to the message file.

Syntax

```
►►──CEEMOUT──(──message_string──,──destination_code──,──fc──)─────────────────►◄
```

message_string **(input)**
A halfword-prefixed printable character string containing the message. DBCS characters must be enclosed within shift-out (byte X'0F') shift-in (X'0E') characters. Insert data cannot be placed in the message with CEEMOUT. The halfword-prefixed message string (input) must contain only printable characters. For length greater than zero, unpredictable results will occur if the byte following the halfword prefix is *X'00"*

destination_code **(input)**
A 4-byte binary integer. The only accepted value for *destination_code* is 2. Under systems other than CICS, Language Environment writes the message to the *ddname* of the file specified in the MSGFILE runtime option. Under CICS, the message is written to a transient data queue named CESE.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0E3	3	0451	An invalid destination code *destination-code* was passed to routine *routine-name*.
CEE0E9	3	0457	The message file destination *ddname* could not be located.

Usage notes

- z/OS UNIX considerations—In multithread applications, CEEMOUT affects only the calling thread. 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).
- If the message file is defined with an LRECL greater than 256, and the input to CEEMOUT is greater than 256 bytes, the output results in multiple records. The first 256 bytes of each record contains output data. The remaniing bytes of each record, up to the LRECL size, might contain unpredictable data.

For more information

- See "MSGFILE" on page 53 for more information about the MSGFILE runtime option.
- See *z/OS Language Environment Programming Guide* for more information about CESE.

Examples

1. Following is an example of CEEMOUT called by C/C++.

```
/*Module/File Name: EDCMOUT   */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _VSTRING message;
  _INT4 dest;
  _FEEDBACK fc;

  strcpy(message.string,"This is a test message");
  message.length = strlen(message.string);
  dest = 2;

  CEEMOUT(&message,&dest,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEMOUT failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  /* .
     .
     . */
}
```

2. Following is an example of CEEMOUT called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTMOUT
      ************************************************
      ** CBLMOUT - Call CEEMOUT to dispatch a msg.  **
      ** In this example, a call is made to CEEMOUT **
      ** to dispatch a user-defined message string  **
      ** to the ddname specified defaulted in the   **
      ** MSGFILE runtime option.                   **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLMOUT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  MSGSTR.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                              of MSGSTR.
       01  DESTIN                  PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
```

```
         02  I-S-Info            PIC S9(9) BINARY.
     PROCEDURE DIVISION.
     PARA-CBLMOUT.
    *************************************************
    ** Create message string and specify length    **
    *************************************************
         MOVE 25 TO Vstring-length of MSGSTR.
         MOVE "CEEMOUT ran successfully"
              TO Vstring-text of MSGSTR.
    *************************************************
    ** Specify 2 to send the message to the ddname **
    **     specified or defaulted in the MSGFILE   **
    **     runtime option.                        **
    *************************************************
         MOVE 2 TO DESTIN.
         CALL "CEEMOUT" USING MSGSTR, DESTIN, FC.
         IF  NOT CEE000 of FC  THEN
             DISPLAY "CEEMOUT failed with msg "
                 Msg-No of FC UPON CONSOLE
             STOP RUN
         END-IF.
         GOBACK.
```

3. Following is an example of CEEMOUT called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMMOUT                          */
 /****************************************************/
 /** Function: CEEMOUT - Dispatch a message         **/
 /**                                                **/
 /** In this example, CEEMOUT is called to dispatch **/
 /** a user-defined message string to the ddname    **/
 /** specified or defaulted in the MSGFILE runtime **/
 /** option.                                        **/
 /****************************************************/
 PLIMOUT: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL MSGSTR CHAR(255) VARYING;
    DCL DESTIN REAL FIXED BINARY(31,0);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    MSGSTR = 'CEEMOUT ran successfully.';
                           /* Set message string     */
    DESTIN = 2;            /* Send to MSGFILE ddname */

    /* Dispatch message to destination by a  */
    /*    call to CEEMOUT                    */
    CALL CEEMOUT ( MSGSTR, DESTIN, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'Message "' || MSGSTR
          || '" sent to destination ' || DESTIN );
       END;
    ELSE  DO;
       DISPLAY( 'CEEMOUT failed with msg '
          || FC.MsgNo );
```

```
           STOP;
           END;

      END PLIMOUT;
```

CEEMRCE—Move resume cursor explicit

The CEEMRCE service resumes execution of a user routine at the location established by CEE3SRP. CEEMRCE is designed to be called from a user condition handler and works only in conjunction with the CEE3SRP service.

Syntax

```
►►──CEEMRCE──(──resume_token──,──fc──)────────────────────────────────►◄
```

resume_token **(input)**
An INT4 data type that contains a token, returned from the CEE3SRP service, representing the resume point in the user routine.

fc **(output/optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE07V	2	0255	The first parameter passed to CEEMRCE was an unrecognized label.

Usage notes

- Exit DSA routines are invoked as the resume cursor is moved back across stack frames.
- When a resume is requested, the state of the machine indicated in the machine state block is established prior to entry at the resume point.

For more information

- See *z/OS Language Environment Programming Guide* for more information about the CEEMRCE callable service.

Examples

1. An example of CEEMRCE called by COBOL:

```
        CBL NODYNAM APOST
        IDENTIFICATION DIVISION.
        PROGRAM-ID. PGM2.
       *Module/File Name: IGZTMRCE
       *__________________________________________*
       *                                          *
       * Sample program using CEE3SRP and CEEMRCE.*
       * PGM2 registers user-written condition    *
       * handler UCH1 using CEEHDLR.  It          *
       * sets a resume point using CEE3SRP. After *
```

```
* incurring a condition and returning      *
* to PGM2, PGM3 is called.  PGM3 sets up   *
* new resume point, does a divide-by-zero, *
* and after resuming in PGM3, resets the   *
* resume point to PGM2 and does a GOBACK.  *
*__________________________________________*

 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  RECOVERY-AREA EXTERNAL.
     05  RECOVERY-POINT            POINTER.
     05  ERROR-INDICATOR           PIC X(01).

 01 UCH-ROUTINE      PROCEDURE-POINTER.

 01  FIELDS.
     05  FIRST-TIME-SW    PIC X(03) VALUE ' ON'.
         88 FIRST-TIME-88           VALUE ' ON'.
     05  ANSWER    PIC S9(02) COMP-3 VALUE 0.
     05  UCH1      PIC  X(08) VALUE 'UCH1    '.
     05  PGM3      PIC  X(08) VALUE 'PGM3    '.
     05  CEEHDLR   PIC  X(08) VALUE 'CEEHDLR '.
     05  CEE3SRP   PIC  X(08) VALUE 'CEE3SRP '.
     05  TOKEN     PIC S9(09) BINARY.
     05  FC.
         10  CASE-1.
             15  SEVERITY PIC S9(04) BINARY.
             15  MSG-NO   PIC S9(04) BINARY.
         10  SEV-CTL      PIC  X(01).
         10  FACILITY-ID  PIC  X(03).
         10  I-S-INFO     PIC S9(09) BINARY.

 PROCEDURE DIVISION.

     SET UCH-ROUTINE TO ENTRY 'UCH1'.
*__________________________________________*
*                                          *
* Register the condition handler, UCH1.    *
*__________________________________________*

     CALL CEEHDLR USING UCH-ROUTINE, TOKEN, FC.
     IF CASE-1 NOT = LOW-VALUE
         GOBACK.
     PERFORM COMPUTE-LOOP 3 TIMES.
     CALL PGM3 USING RECOVERY-AREA.
     SET RECOVERY-POINT TO NULL.
     GOBACK.       COMPUTE-LOOP.
     IF FIRST-TIME-88
         MOVE 'OFF' TO FIRST-TIME-SW
*__________________________________________*
*                                          *
* Set up a new resume point.               *
*__________________________________________*

         CALL CEE3SRP USING RECOVERY-POINT,
                            CASE-1

                            SERVICE LABEL

         IF CASE-1 NOT = LOW-VALUE
             GOBACK.

     IF ERROR-INDICATOR = 'E'
         MOVE SPACE TO ERROR-INDICATOR
         MOVE 1 TO ANSWER.
```

```
* Application code may go here.

     COMPUTE ANSWER = 1 / ANSWER.

* Put application code here.

 CBL NODYNAM APOST
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PGM3.
*_________________________________________*
*                                         *
* Sample program using CEE3SRP and CEEMRCE.*
* PGM2 registered UCH1. This program sets a*
* new resume point, does a divide-by-zero, *
* and after resuming in PGM3, resets the   *
* resume point to PGM2 and does a GOBACK.  *
*_________________________________________*

 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  RECOVERY-AREA EXTERNAL.
     05  RECOVERY-POINT            POINTER.
     05  ERROR-INDICATOR           PIC X(01).

 01 UCH-ROUTINE      PROCEDURE-POINTER.      01  FIELDS.
     05  FIRST-TIME-SW    PIC X(03) VALUE ' ON'.
         88 FIRST-TIME-88           VALUE ' ON'.
     05  ANSWER    PIC S9(02) COMP-3 VALUE 0.
     05  CEEHDLR   PIC  X(08) VALUE 'CEEHDLR '.
     05  CEE3SRP   PIC  X(08) VALUE 'CEE3SRP '.
     05  TOKEN     PIC S9(09) BINARY.
     05  SEV PIC -9(05).
     05  MSG PIC -9(05).
     05  FC.
         10  CASE-1.
             15  SEVERITY PIC S9(04) BINARY.
             15  MSG-NO   PIC S9(04) BINARY.
         10  SEV-CTL      PIC  X(01).
         10  FACILITY-ID  PIC  X(03).
         10  I-S-INFO     PIC S9(09) BINARY.


 PROCEDURE DIVISION.

     PERFORM COMPUTE-LOOP 3 TIMES.
     SET RECOVERY-POINT TO NULL.
     GOBACK.

 COMPUTE-LOOP.
     IF FIRST-TIME-88
         MOVE 'OFF' TO FIRST-TIME-SW
*_________________________________________*
*                                         *
* Set new resume point.                   *
*_________________________________________*

         CALL CEE3SRP USING RECOVERY-POINT, FC

         SERVICE LABEL

         IF CASE-1 NOT = LOW-VALUE
             GOBACK.

     IF ERROR-INDICATOR = 'E'
         MOVE SPACE TO ERROR-INDICATOR
         MOVE 1 TO ANSWER.
```

```
      * Application code may go here.

           COMPUTE ANSWER = 1 / ANSWER.

      * Put application code here.

       CBL NODYNAM APOST
       IDENTIFICATION DIVISION.
       PROGRAM-ID. UCH1.
*__________________________________________*
      *                                         *
      * Sample user condition handler using     *
      * CEEMRCE.  This program sets an error    *
      * flag for the program-in-error to query  *
      * and issues a call to CEEMRCE to return  *
      * control to the statement following the  *
      * call to CEE3SRP.                        *
      *_________________________________________*
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  RECOVERY-AREA EXTERNAL.
           05 RECOVERY-POINT         POINTER.
           05 ERROR-INDICATOR        PIC X(01).

       01  FC.
           10  CASE-1.
               15  SEVERITY PIC S9(04) BINARY.
               15  MSG-NO   PIC S9(04) BINARY.
           10  SEV-CTL      PIC  X(01).
           10  FACILITY-ID  PIC  X(03).
           10  I-S-INFO     PIC S9(09) BINARY.

       01 CEEMRCE       PIC X(08) VALUE 'CEEMRCE '.
       LINKAGE SECTION.

       01  CURRENT-CONDITION         PIC  X(12).
       01  TOKEN                     PIC  X(04).
       01  RESULT-CODE             PIC S9(09) BINARY.
           88  RESUME                 VALUE +10.
           88  PERCOLATE              VALUE +20.
           88  PERC-SF                VALUE +21.
           88  PROMOTE                VALUE +30.
           88  PROMOTE-SF             VALUE +31.
       01  NEW-CONDITION             PIC  X(12).

       PROCEDURE DIVISION  USING CURRENT-CONDITION,
                                 TOKEN,
                                 RESULT-CODE,
                                 NEW-CONDITION.

           MOVE 'E' TO ERROR-INDICATOR.

      *_________________________________________*
      *                                         *
      * Call CEEMRCE to return control to the   *
      * last resume point.                      *
      *_________________________________________*

           CALL CEEMRCE USING RECOVERY-POINT,
                              FC.
           IF CASE-1 NOT = LOW-VALUE
```

```
                GOBACK.
            MOVE +10 TO RESULT-CODE.

            GOBACK.
```

2. An example of CEEMRCE called by PL/I:

```
*Process lc(101),opt(0),s,map,list,stmt,a(f),ag;
*Process macro;
 DRV3SRP: Proc Options(Main);

   /*Module/File Name: IBM3SRP                       */
   /****************************************************
    **                                               *
    ** DRV3SRP - Set an explicit resume point by      *
    **           calling CEE3SRP then registering a   *
    **           condition handler that calls CEEMRCE *
    **           to resume at the explicitly set      *
    **           resume point.                        *
    **                                               *
    **************************************************/

    %include CEEIBMCT;
    %include CEEIBMAW;
    declare 01 FBCODE   feedback;  /* Feedback token */
    declare DENOMINATOR fixed binary(31,0);
    declare NUMERATOR   fixed binary(31,0);
    declare RATIO       fixed binary(31,0);
    declare PLI3SRP     external entry;
    declare U_PTR       pointer;
    declare 01 U_DATA,
              03 U_CNTL fixed binary(31),
              03 U_TOK  pointer;

         U_PTR = addr(U_DATA);
         U_CNTL = 0;

         /* Set Resume Point                */

         Display('Setting resume point via CEE3SRP');
         Call CEE3SRP(U_TOK,FBCODE);
         Display('After CEE3SRP ... Resume point');
         If U_CNTL = 0 Then
           Do;
             Display('First time through...');

             Display('Registering user handler');
             Call CEEHDLR(PLI3SRP, U_PTR, FBCODE);
             If FBCHECK(FBCODE, CEE000) Then
               Do;
                 /* Cause a zero-divide condition  */

                 DENOMINATOR = 0;
                 NUMERATOR = 1;
                 RATIO = NUMERATOR / DENOMINATOR;
               End;
             Else
               Do;
                 Display('CEEHDLR failed with msg ');
                 Display(MsgNo);
               End;
           End;
         Else
           Display('Second time through...');

         /* Unregister handler                */
Call CEEHDLU(PLI3SRP, FBCODE);
         If FBCHECK(FBCODE, CEE000) Then
```

```
          Display('Main: unregistered PLI3SRP');
        Else
          Do;
            Display('CEEHDLU failed with msg ');
            Display(MsgNo);
          End;
 End DRV3SRP;

*Process lc(101),opt(0),s,map,list,stmt,a(f),ag;
*Process macro;
 PLI3SRP: Proc (PTR1,PTR2,PTR3,PTR4) Options(byvalue);
   /*************************************************
    **                                              *
    ** PLI3SRP - Call CEEMCRE to resume at the resume *
    **           point explicitly set in user        *
    **           program.                            *
    **                                              *
    *************************************************/

    %include CEEIBMCT;
    %include CEEIBMAW;
    declare (PTR1,PTR2,PTR3,PTR4) pointer;
    declare 01 CURCOND  based(PTR1) feedback;
    declare TOKEN       pointer based(PTR2);
    declare RESULT      fixed bin(31,0) based(PTR3);
    declare 01 NEWCOND  based(PTR4) feedback;
    declare 01 U_DATA   based(TOKEN),
              03 U_CNTL fixed binary(31,0),
              03 U_TOK  pointer;
    declare 01 FBCODE   feedback;

    Display('In user handler');
    RESULT = 10;
    Call CEEMRCE(U_TOK,FBCODE);
    Display(U_CNTL);
    U_CNTL = 1;
    Return;
```

CEEMRCR—Move resume cursor

CEEMRCR moves the resume cursor to a position relative to the current position of the handle cursor. The actions supported are:

- Moving the resume cursor to the call return point of the routine registering the executing condition handler.
- Moving the resume cursor to the caller of the routine registering the executing condition handler.

Initially, the resume cursor is placed after the instruction that caused the condition. Whenever CEEMRCR moves the resume cursor and passes stack frames, associated exit routines are invoked. Note that "exit routine" refers to user condition handlers as well as language-specific condition handlers. In addition, any associated user condition handlers are unregistered. The movement direction is always toward earlier stack frames, never toward more recent stack frames. The movement occurs only after the condition handler returns to the Language Environment condition manager.

Multiple calls to CEEMRCR yield the net results of the calls; that is, if two calls move the resume cursor to different places for the same stack frame, the most restrictive call (that closest to the earliest stack frame) is used for that stack frame.

Moving the resume cursor to a particular stack frame:

- Cancels all stack frames from the previous resume point up to but not including the new resume point
- Unregisters any user condition handlers registered for the canceled stack frames

Syntax

```
►►──CEEMRCR──(──type_of_move──,──fc──)──────────────────────────────────►◄
```

type_of_move **(input)**
A fullword binary signed integer indicating the target of the resume cursor movement. The possible values for *type_of_move* are:

0 Move the resume cursor to the call return point of the stack frame associated with the handle cursor.

1 Move the resume cursor to the call return point of the stack frame prior to the stack frame associated with the handle cursor. The handle cursor is moved to the most recently established condition handler of the stack frame. The new resume cursor position now points to this condition handler for the stack frame. Do not use a *type_of_move* value of 1 if the caller of the stack frame associated with the handle cursor is a nested COBOL program.

Modifying the resume cursor to point to stack frame 0 is not allowed. You cannot move the resume cursor beyond the earliest stack frame.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE07U	1	0254	The first parameter passed to CEEMRCR was not 0 or 1.
CEE083	3	0259	A move to stack frame zero using CEEMRCR was attempted from a MAIN routine.
CEE084	3	0260	No condition was active when a call to a condition management routine was made. The requested function was not performed.
CEE08L	1	0277	CEEMRCR was called to perform an unnecessary move.

Usage notes

- PL/I MTF consideration—CEEMRCR is not supported in PL/I MTF applications. This includes any CEEHDLR service called from a COBOL program in the application.
- z/OS UNIX considerations—In multithread applications, CEEMRCR affects only the calling thread. You can use CEEMRCR only within the thread's call chain.

Illustration of CEEMRCR Usage

The following three figures illustrate how you can move the resume cursor by using the CEEMRCR service.

In Figure 9, routine A calls routine B, which in turn calls C, which calls D. User condition handlers are registered in routines B and C.

When a condition is raised in routine D, the Language Environment condition manager passes control to the user condition handler established for routine C. The handle cursor now points to the stack frame for routine C. Routine C percolates the condition.

The handle cursor now points to the stack frame for routine B. The next user condition handler to gain control is that one established for routine B; it recognizes the condition and issues a resume by calling CEEMRCR.

A 0 *type_of_move*, meaning move the resume cursor to the stack frame associated with the handle cursor, causes control to resume at the call return point in routine B, the instruction immediately following the call to routine C. A 1 *type_of_move*, meaning move the resume cursor to the call return point of the stack frame immediately preceding the one to which the handle cursor points, moves the resume cursor to the instruction immediately following a call in routine A to routine B.

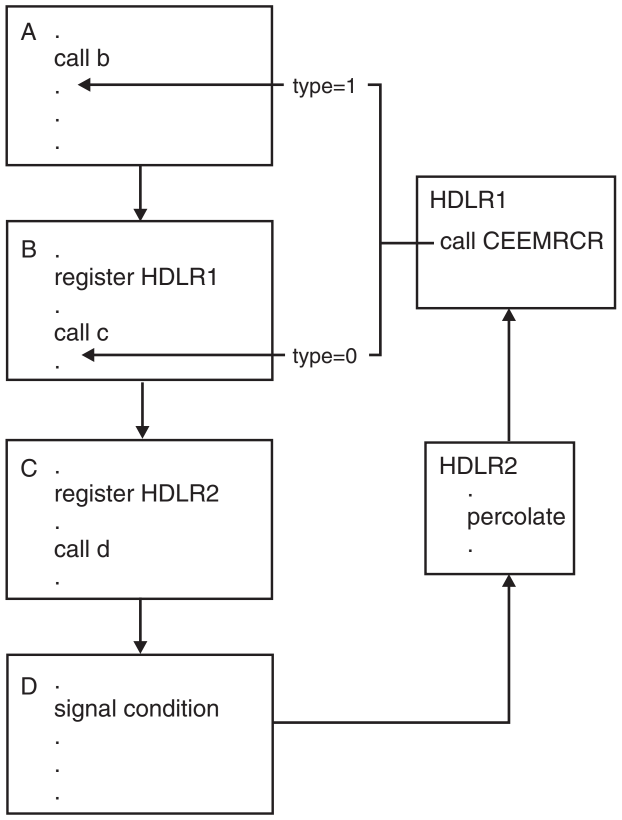


Figure 9. First example moving resume cursor using CEEMRCR

The same scenario is illustrated in Figure 10 on page 366, except that HDLR2 issues a resume for the signaled condition rather than percolating it. HDLR1 never gains control. Because the handle cursor now points to the stack frame for routine C, a 0 *type_of_move* causes control to resume at the call return point in routine C, the instruction immediately following the call to routine D. A 1 *type_of_move* moves the resume cursor to the instruction immediately following a call in routine B to routine C.

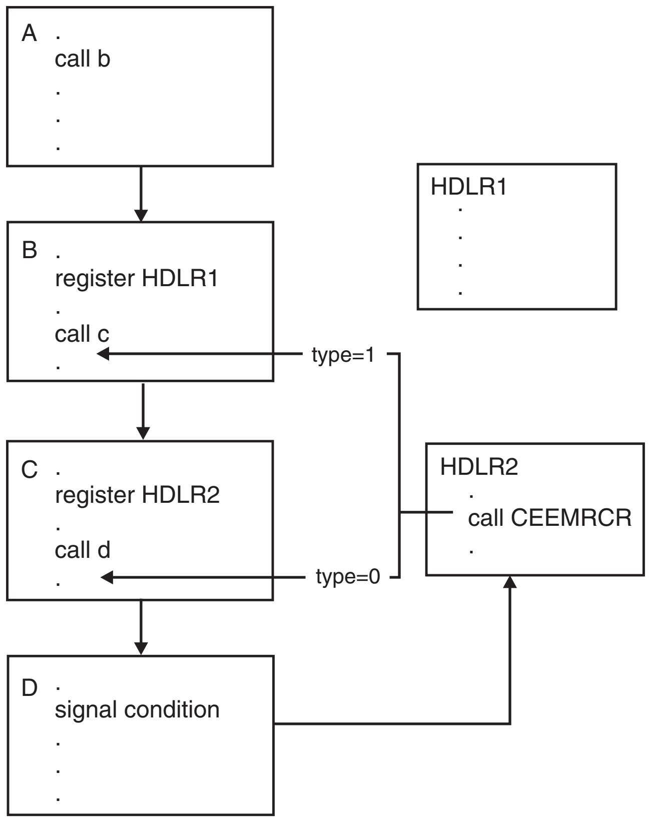


Figure 10. Second example moving resume cursor using CEEMRCR

In Figure 11 on page 367, the user condition handlers are established for routines C and D. When a condition is raised in routine D, only a 1 *type_of_move* is permitted. A 0 *type_of_move* results in error warning message CEE0277.

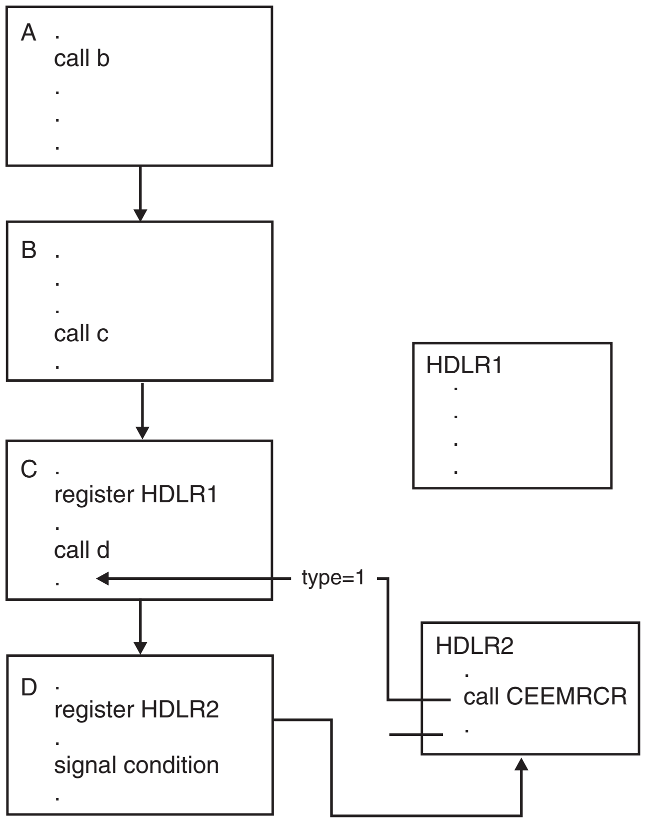


Figure 11. Third example moving resume cursor using CEEMRCR

Examples

1. Following is an example of CEEMRCR called by C/C++.

```
/*Module/File Name: EDCMRCR   */

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

#ifdef __cplusplus
extern "C" {
#endif
void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
}
#endif

void b(void);

int main(void) {
/*

⋮
 */
  b();
  /* the CEEMRCR call in the handler will place the */
  /* resume cursor at this point.  */
/*
```

```
⋮
 */
}

void b(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;

  /* register the condition handler */
  token = 99;
  routine.address = (_POINTER)&handler;;
  routine.nesting = NULL;
  CEEHDLR(&routine,&token,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEEHDLR failed with message number %d\n",
            fc.tok_msgno);
    exit (2999);
  }
/*

⋮
 */

 /* set up the condition using CEENCOD */
  c_1 = 3;
  c_2 = 2523;
  cond_case = 1;
  sev = 3;
  control = 0;
  memcpy(facid,"CEE",3);
  isi = 0;

  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&condtok,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  /* signal the condition */
  CEESGL(&condtok,&qdata,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEESGL failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
/*
⋮
*/
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
             _FEEDBACK *newfc) {

   _FEEDBACK cursorfc, orig_fc;
   _INT4 type;
/*  .
    .
    . */
   /* move the resume cursor to the caller of the  */
   /* routine that registered the condition handler */
   type = 1;
   CEEMRCR(&type,&cursorfc);
   if ( _FBCHECK ( cursorfc , CEE000 ) != 0 ) {
      printf("CEEMRCR failed with message number %d\n",
              cursorfc.tok_msgno);
      exit (2999);
   }
   printf("condition handled\n");
   *result = 10;
   return;
}
```

2. Following is an example of CEEMRCR called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTMRCR
      ************************************************
      **                                            **
      ** CBLMAIN - Main for sample program for      **
      **           CEEMRCR.                         **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLMAIN.
       PROCEDURE DIVISION.
           CALL "DRVMRCR"
           DISPLAY "Resumed execution in the CALLER "
               "of the routine which registered the "
               "handler"
           GOBACK.
       END PROGRAM CBLMAIN.

      ************************************************
      **                                            **
      ** DRVMRCR - Drive sample program CEEMRCR.    **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRVMRCR.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE             PROCEDURE-POINTER.
       01  TOKEN               PIC S9(9) BINARY.
       01  SEV                 PIC S9(4) BINARY.
       01  MSGNO               PIC S9(4) BINARY.
       01  CASE                PIC S9(4) BINARY.
       01  SEV2                PIC S9(4) BINARY.
       01  CNTRL               PIC S9(4) BINARY.
       01  FACID               PIC X(3).
       01  ISINFO              PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       01  QDATA                   PIC S9(9) BINARY.
       01  CONDTOK.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
```

```
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION.
************************************************
** Register handler                           **
************************************************
     SET ROUTINE TO ENTRY "CBLMRCR".
     CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
     IF  NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLR failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

************************************************
** Signal a condition                         **
************************************************
     MOVE 1 TO QDATA.
     SET CEE001 of CONDTOK to TRUE.
     MOVE ZERO to I-S-Info of CONDTOK.
     CALL "CEESGL" USING CONDTOK, QDATA, FC.
     IF CEE000 of FC  THEN
         DISPLAY "**** Resumed execution in the "
           "routine which registered the handler"
     ELSE
         DISPLAY "CEESGL failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.
************************************************
** UNregister handler                         **
************************************************
     CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEEHDLU failed with msg "
             Msg-No of FC UPON CONSOLE
     END-IF.
     STOP RUN.
 END PROGRAM DRVMRCR.
************************************************
**                                            **
** CBLMRCR - Invoke CEEMRCR to Move resume    **
**           cursor relative to handle cursor **
**                                            **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLMRCR.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  MOVETYP           PIC S9(9) BINARY.
 01  DEST              PIC S9(9) BINARY VALUE 2.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
```

```
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    01  FC2.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    LINKAGE SECTION.
    01  CURRENT-CONDITION.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    01  TOKEN                   PIC X(4).
    01  RESULT-CODE             PIC S9(9) BINARY.
        88  RESUME              VALUE +10.
        88  PERCOLATE           VALUE +20.
        88  PROMOTE             VALUE +30.
    01  NEW-CONDITION.
        02  Condition-Token-Value.
        COPY  CEEIGZCT.
            03  Case-1-Condition-ID.
                04  Severity    PIC S9(4) BINARY.
                04  Msg-No      PIC S9(4) BINARY.
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    PROCEDURE DIVISION USING CURRENT-CONDITION,
                             TOKEN, RESULT-CODE,
                             NEW-CONDITION
   ************************************************
   ** Move the resume cursor to the caller of    **
   ** the routine that registered the condition  **
   ** handler                                    **
   ************************************************
        MOVE 1 TO MOVETYP.
        CALL "CEEMRCR" USING MOVETYP , FC.
        IF  NOT CEE000 of FC  THEN
            DISPLAY "CEEMRCR failed with msg "
                Msg-No of FC UPON CONSOLE
            CALL "CEEMSG" USING FC, DEST, FC2
            IF  NOT CEE000 of FC2  THEN
              DISPLAY "CEEMSG failed with msg "
                  Msg-No of FC2 UPON CONSOLE
            MOVE FC TO NEW-CONDITION
```

```
                SET PROMOTE TO TRUE
                GOBACK
            END-IF.

            SET RESUME TO TRUE.

            GOBACK.

        END PROGRAM CBLMRCR.
```

3. Following is an example of a PL/I to handle divide-by-zero condition.

```
*Process macro;
 /* Module/File Name: IBMMRCR                        */
 /****************************************************/
 /*                                                  */
 /* Usrhdlr - the user handler routine.              */
 /* Handle DIVIDE-BY-ZERO conditions,                */
 /*     percolate all others.                        */
 /*                                                  */
 /****************************************************/
 Usrhdlr:  Proc (@condtok, @token, @result, @newcond)
           options(byvalue);

   %include ceeibmct;
   %include ceeibmaw;

   /* Parameters */
   dcl  @condtok    pointer;
   dcl  @token      pointer;
   dcl  @result     pointer;
   dcl  @newcond    pointer;
   dcl 1 condtok based(@condtok) feedback;
   dcl token  fixed bin(31) based(@token);
   dcl result fixed bin(31) based(@result);
   dcl 1 newcond based(@newcond) feedback;
   dcl 1 fback feedback;
   dcl move_type fixed bin(31);
   dcl resume     fixed bin(31) static initial(10);
   dcl percolate  fixed bin(31) static initial(20);
   dcl promote    fixed bin(31) static initial(30);
   dcl promote_sf fixed bin(31) static initial(31);
   display ('>>> USRHDLR: Entered user handler');
   display ('>>> USRHDLR: passed token value is ' ||
            token);

   /* Check if this is the divide-by-zero token */
   if fbcheck (condtok, cee349) then
     do;
       move_type = 0;
       call ceemrcr (move_type, fback);
       If fbcheck (fback, cee000) then
         do;
           result = resume;
           display ('>>> USRHDLR: Resuming execution');
         end;
       else
         do;
           display
           ('CEEMRCR failed with message number ' ||
             fback.MsgNo);
           stop;
         end;
     end;
   else     /* something besides div-zero token */
     do;
```

```
          result = percolate;
          display ('>>> USRHDLR: Percolating it');
        end;
    end Usrhdlr;
```

CEEMSG—Get, format, and dispatch a message

CEEMSG gets, formats, and dispatches a message corresponding to an input condition token received from a callable service or passed to a user-written condition handler. You can use this service to print a message after a call to any Language Environment service that returns a condition token.

Syntax

```
►►──CEEMSG──(──cond_token──,──destination_code──,──fc──)──────────────────►◄
```

cond_token **(input)**
A 12-byte condition token received as the result of a Language Environment callable service.

destination_code **(input)**
A 4-byte binary integer. *destination_code* can be specified only as 2, meaning write the message to the *ddname* of the file specified in the MSGFILE runtime option .

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0E2	3	0450	The message inserts for the condition token with message number *message-number* and facility ID *facility-id* could not be located.
CEE0E3	3	0451	An invalid destination code *destination-code* was passed to routine *routine-name*.
CEE0E6	3	0454	The message number *message-number* could not be found for facility ID *facility-id*.
CEE0E9	3	0457	The message file destination *ddname* could not be located.
CEE0EA	3	0458	The message repository *repository-name* could not be located.
CEE3CT	3	3485	An internal message services error occurred while locating the message number within a message file.
CEE3CU	3	3486	An internal message services error occurred while formatting a message.
CEE3CV	3	3487	An internal message services error occurred while locating a message number within the ranges specified in the repository.

Usage notes

- z/OS UNIX considerations—In multithread applications, CEEMSG affects only the calling thread. 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).

For more information

- See "CEENCOD—Construct a condition token" on page 377 for more information about the CEENCOD callable service.
- See "MSGFILE" on page 53 for more information about the MSGFILE runtime option.

Examples

1. Following is an example of CEEMSG called by C/C++.

```
/*Module/File Name: EDCMSG    */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _VSTRING message;
  _INT4 dest,msgindx;
  _CHAR80 msgarea;
  _FEEDBACK fc,token;

  strcpy(message.string,"This is a test message");
  message.length = strlen(message.string);
  dest = 5; /* invalid dest so CEEMOUT will fail */

  CEEMOUT(&message,&dest,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     /* put the message if CEEMOUT failed */
     dest = 2;
     CEEMSG(&fc,&dest,NULL);
     exit(2999);
  }
}
```

2. Following is an example of CEEMSG called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTMSG
      *************************************************
      **                                             **
      ** CBLMSG - Call CEEMSG to get, format and     **
      **                    dispatch a message       **
      **                                             **
      ** In this example, CEE3MDS is called with an  **
      ** invalid country code so that a condition    **
      ** token would be returned to use as input to  **
      ** Any Lang Env service could have been called.**
      ** CEEMSG uses the condition token to get,     **
      ** format and dispatch the message associated  **
      ** with the condition that occurred in CEE3MDS.**
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLMSG.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COUNTRY                 PIC X(2).
       01  DECSEP                  PIC X(2).
       01  MSGDEST                 PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       01  FC2.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBL3MDS.

     *************************************************
      ** Call a Lang Env svc, CEE3MDS in this case,  **
      ** to receive a condition token that CEEMSG    **
      ** can format as a message.  Specify an        **
      ** invalid value for country code so that a    **
      ** condition will be built                     **
      *************************************************
           MOVE "LN" TO COUNTRY.
           CALL "CEE3MDS" USING COUNTRY, DECSEP, FC.
       PARA-CBLMSG.
      *************************************************
      ** Specify 2 for destination, so message will  **
      ** be written to the ddname specified or       **
      ** defaulted in the MSGFILE runtime option.   **
      *************************************************
           MOVE 2 TO MSGDEST.
      *************************************************
      ** Call CEEMSG using the FC returned from      **
      **     CEE3MDS as the input condition token.   **
      *************************************************
           CALL "CEEMSG" USING FC, MSGDEST, FC2.
           IF NOT CEE000 of FC2 THEN
               DISPLAY "CEEMSG  failed with msg "
                   Msg-No of FC2 UPON CONSOLE
               STOP RUN
           END-IF.
           GOBACK.
```

3. Following is an example of CEEMSG called by PL/I.

```
*PROCESS LANGLVL(SAA), MACRO;
 /* Module/File Name: IBMMSG                        */
 /**************************************************/
 /**                                              **/
 /** Function: CEEMSG  - get, format and dispatch   **/
/**                    a message                  **/
/**                                               **/
/** In this example, CEE3MDS is called with an     **/
/** invalid country code so that a condition token **/
/** would be returned to use as input to CEEMSG.   **/
/** Any LE/370 could have been called. CEEMSG uses **/
/** the condition to get, format and dispatch the  **/
/** message associated with the condition that     **/
/** occurred in CEE3MDS                            **/
/**                                               **/
/*************************************************/
PLIMSG: PROC OPTIONS(MAIN);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL COUNTRY  CHARACTER ( 2 );
   DCL DECSEP   CHARACTER ( 2 );
   DCL 01 FC,                     /* Feedback token */
          03 MsgSev    REAL FIXED BINARY(15,0),
          03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),      /* Facility ID */
          03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);
   DCL MSGDEST  REAL FIXED BINARY(31,0);
   DCL 01 FC2,                    /* Feedback token */
          03 MsgSev    REAL FIXED BINARY(15,0),
          03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),      /* Facility ID */
          03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);

   COUNTRY = 'LN';  /* Specify an invalid country   */
                    /* code to receive a non-zero   */
                    /* feedback code                */

   /* Call any service (CEE3MDS in this case) to    */
   /* receive a condition token that CEEMSG will    */
   /* format and dispatch a message                 */
   CALL CEE3MDS ( COUNTRY, DECSEP, FC );

   MSGDEST = 2; /* Specify 2 as destination, so     */
                /* message will go to ddname speci- */
                /* fied in MSGFILE runtime option  */

   CALL CEEMSG ( FC, MSGDEST, FC2 );
   IF  ¬ FBCHECK( FC2, CEE000)  THEN  DO;
      DISPLAY( 'CEEMSG  failed with msg '
         || FC.MsgNo );
      STOP;
      END;

END PLIMSG;
```

CEENCOD—Construct a condition token

CEENCOD dynamically constructs a 12-byte condition token that communicates a condition in Language Environment. The condition token communicates with the Language Environment message and condition handling callable services, and user routines. Also, all Language Environment callable services use the condition-token data type to return information to the user as a feedback code.

Syntax

```
►►──CEENCOD──(──c_1──,──c_2──,──case──,──severity──,──control──,──────────────►

►─facility_ID──,──i_s_info──,──cond_token──,──fc──)──────────────────────────►◄
```

c_1 **(input)**
c_1 and *c_2* together make up the condition_ID portion of the condition token. *c_1* is a 2-byte binary integer representing the value of the first 2 bytes of the 4-byte condition_ID. For case 1, *c_1* represents the severity; for case 2, it is the class_code.

c_2 **(input)**
A 2-byte binary integer representing the value of the second 2 bytes of the condition_ID. For case 1, this is the Msg_No; for case 2, it is the cause_code.

case **(input)**
A 2-byte binary integer defining the format of the condition_ID portion of the token.

severity **(input)**
A 2-byte binary integer indicating the condition's severity. For case 1 conditions, the value of this field is the same as the severity value specified in the condition_ID. For case 1 and 2 conditions, this field is also used to test the condition's severity. *severity* can be specified with the following values:

0	Information only (or, if the entire token is 0, no information).
1	Warning; service completed, probably correctly.
2	Error detected and correction attempted; service completed, perhaps incorrectly.
3	Severe error; service not completed.
4	Critical error; service not completed; condition signaled. A critical error is a condition jeopardizing the environment. If a critical error occurs during a Language Environment callable service, it is always signaled to the condition manager instead of returning synchronously to the caller.

control **(input)**
A 2-byte binary integer containing flags describing or controlling various aspects of condition handling. Valid values for the control field are 1 and 0. 1 indicates the *facility_ID* is assigned by IBM. 0 indicates the *facility_ID* is assigned by the user.

facility_ID **(input)**
A 3-character field containing three alphanumeric characters (A-Z, a-z and 0-9)

identifying the product or component of a product generating this condition or feedback information, for example, CEE.

The *facility_ID* is associated with the repository (for example, a file) of the runtime messages. If a unique ID is required (for IBM and non-IBM products), an ID can be obtained by contacting an IBM project office.

If you create a new *facility_ID* to use with a message file you created by using the CEEBLDTX utility, be aware that the *facility_ID* must be part of the message file name. It is therefore important to follow the naming guidelines described below in order to have a module name that does not cause your application to abend.

Begin a non-IBM assigned product *facility_ID* with letters J through Z. (See the preceding description *control* (input) parameter on how to indicate whether the *facility_ID* has been assigned by IBM.) Special characters, including blank spaces, cannot be used in a *facility_ID*. There are no other constraints (besides the alphanumeric requirement) on a non-IBM assigned *facility_ID*.

i_s_info **(input)**
A fullword binary integer identifying the ISI, that contains insert data. Whenever a condition is detected by the Language Environment condition manager, insert data is generated describing the instance of its occurrence is generated. This insert data is used, for example, to write to a file a message associated with that instance or occurrence of the condition.

cond_token **(output)**
The 12-byte representation of the constructed condition token.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE0CH	3	0401	An invalid case code *case-code* was passed to routine *routine-name*.
CEE0CI	3	0402	An invalid control code *control-code* was passed to routine *routine-name*.
CEE0CJ	3	0403	An invalid severity code *severity-code* was passed to routine *routine-name*.
CEE0CK	1	0404	Facility ID *facility-id* with non-alphanumeric characters was passed to routine *routine-name*.
CEE0E4	3	0452	An invalid facility ID *facility-id* was passed to routine *routine-name*.

Usage notes

- C/C++ considerations—The structure of the condition token (type_FEEDBACK) is described in the `leawi.h` header file shipped with Language Environment. You can assign values directly to the fields of the token in the header file without using the CEENCOD service.

  Figure 12 on page 379 shows the layout of the type_FEEDBACK condition token in the header file.

```
typedef struct {
   short     tok_sev     ; /* severity            */
   short     tok_msgno   ; /* message number      */
   int       tok_case :2,  /* flags-case/sev/cont */
             tok_sever:3,
             tok_ctrl :3 ;
   char      tok_facid[3]; /* fac ID    */
   int       tok_isi     ; /* index in ISI block  */
        }                _FEEDBACK;
```

Figure 12. type_FEEDBACK data type as defined in the leawi.h header file

- z/OS UNIX consideration—In multithread applications, CEENCOD affects only the calling thread.

For more information

- For more information about case 1 and case 2, see CEENCOD “Usage notes” on page 378.

Examples

1. Following is an example of CEENCOD called by C/C++.

```
/*Module/File Name: EDCNCOD   */

/*********************************************************/
/* Note that it is not necessary to use this service.    */
/* The fields may be manipulated directly.               */
/*********************************************************/

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc,condtok;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;

  c_1 = 1;
  c_2 = 99;
  cond_case = 1;
  sev = 1;
  control = 0;
  memcpy(facid,"ZZZ",3);
  isi = 0;

  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&condtok,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
 /* .
    .
    . */
}
```

2. Following is an example of CEENCOD called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTNCOD
      *************************************************
      **                                             **
      ** CBLNCOD - Call CEENCOD to construct a       **
      ** condition token                             **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLNCOD.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  SEV                     PIC S9(4) BINARY.
       01  MSGNO                   PIC S9(4) BINARY.
       01  CASE                    PIC S9(4) BINARY.
       01  SEV2                    PIC S9(4) BINARY.
       01  CNTRL                   PIC S9(4) BINARY.
       01  FACID                   PIC X(3).
       01  ISINFO                  PIC S9(9) BINARY.
       01  NEWTOK.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
       PARA-CBLNCOD.
      *************************************************
      ** Set severity portion of Condition-ID to 0,  **
      **     or information only.                    **
      ** Set msg number portion of Condition-ID to 1.**
      ** Set case to 1. This is a service condition. **
      ** Set severity to 0, for information only.    **
      ** Set control to 1, for Facility-ID has been  **
      **     assigned by IBM.                        **
      ** Set Facility-ID to CEE for a Language       **
      **     Environment condition token.            **
      ** Set I-S-Info to 0, indicating that no       **
      **     Instance Specific Information (ISI) is  **
      **     to be supplied.                         **
      *************************************************
           MOVE 0 TO SEV.
           MOVE 1 TO MSGNO.
           MOVE 1 TO CASE.
           MOVE 0 TO SEV2.
           MOVE 1 TO CNTRL.
           MOVE "CEE" TO FACID.
```

```
          MOVE 0 TO ISINFO.

     *************************************************
     ** Call CEENCOD with the values assigned above **
     ** to build a condition token "NEWTOK"         **
     *************************************************
          CALL "CEENCOD" USING SEV, MSGNO, CASE, SEV2,
                               CNTRL, FACID, ISINFO,
                               NEWTOK, FC.
          IF NOT CEE000 of FC  THEN
              DISPLAY "CEENCOD failed with msg "
                  Msg-No of FC UPON CONSOLE
              STOP RUN
          END-IF.

          GOBACK.
```

3. Following is an example of CEENCOD called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMNCOD                           */
 /****************************************************/
 /**                                                  */
 /** Function: CEENCOD - construct a condition token */
 /**                                                  */
 /****************************************************/
 PLINCOD: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL SEV        REAL FIXED BINARY(15,0);
    DCL MSGNO      REAL FIXED BINARY(15,0);
    DCL CASE       REAL FIXED BINARY(15,0);
    DCL SEV2       REAL FIXED BINARY(15,0);
    DCL CNTRL      REAL FIXED BINARY(15,0);
    DCL FACID      CHARACTER ( 3 );
    DCL ISINFO     REAL FIXED BINARY(31,0);
    DCL 01 NEWTOK,                /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    SEV = 0;       /* Set severity portion of       */
                   /* Condition_ID to 0, or         */
                   /* information only.             */
    MSGNO = 1;     /* Set msg number portion of     */
                   /* Condition_ID to 1.            */
    CASE = 1;      /* Set case to 1. This is a      */
                   /* service condition.            */
    SEV2 = 0;      /* Set severity to 0, or         */
                   /* information only.             */
    CNTRL = 0;     /* Set control to 0, or Facility  */
```

```
                     /* ID has been assigned by user   */
    FACID = 'USR'; /* Set Facility_ID to USR for a   */
                     /* user condition token.          */
    ISINFO = 0;    /* Set I_S_Info to 0, indicating  */
                     /* that no Instance Specific      */
                     /* Information is to be supplied. */
    EENCOD ( SEV, MSGNO, CASE, SEV2,
    CALL CEENCOD ( SEV, MSGNO, CASE, SEV2,
       CNTRL, FACID, ISINFO, NEWTOK, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'CEENCOD created token for msg '
          || NEWTOK.MsgNo || ' and facility '
          || NEWTOK.FacID );
       END;
    ELSE  DO;
       DISPLAY( 'CEENCOD failed with msg '
          || FC.MsgNo );
       STOP;
       END;
 END PLINCOD;
```

CEEQCEN—Query the century window

CEEQCEN queries the century in which Language Environment contains the 2-digit year value. When you want to change the setting, use CEEQCEN to get the setting and then use CEESCEN to save and restore the current setting.

Syntax

```
▸▸──CEEQCEN──(──century_start──,──fc──)─────────────────────────────▸◂
```

century_start **(output)**
: An integer between 0 and 100 indicating the year on which the century window is based. For example, if the Language Environment default is in effect, all 2-digit years lie within the 100-year window starting 80 years prior to the system date. CEEQCEN then returns the value 80. An 80 value indicates to Language Environment that, in 1995, all 2-digit years lie within the 100-year window starting 80 years before the system date (between 1915 and 2014, inclusive).

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Usage notes

- z/OS UNIX considerations—CEEQCEN applies to the enclave, as does the century window.

For more information

- See "CEESCEN—Set the century window" on page 394 for more information about the CEESCEN callable service.

Examples

1. Following is an example of CEEQCEN called by C/C++.

```
/*Module/File Name: EDCQCEN   */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main (void) {

  _INT4 century_start;
  _FEEDBACK fc;

  /* query the century window */
  CEEQCEN(&century_start,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEEQCEN failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  /* if the century window is not 50 set it to 50 */
  if (century_start != 50) {
     century_start = 50;

     CEESCEN(&century_start,&fc);
     if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEESCEN failed with message number %d\n",
               fc.tok_msgno);
        exit(2999);
     }
  }
}
```

2. Following is an example of CEEQCEN called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTQCEN
      *************************************************
      ** CBLQCEN - Call CEEQCEN to query the Lang Env**
      **          century window                     **
      ** In this example, CEEQCEN is called to query **
      ** the date at which the century window starts **
      ** The century window is the 100-year window   **
      ** within which Lang Envir                    **
      ** assumes all two-digit years lie.            **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLQCEN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  STARTCW                PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
```

```
                 04  Class-Code  PIC S9(4) BINARY.
                 04  Cause-Code  PIC S9(4) BINARY.
             03  Case-Sev-Ctl    PIC X.
             03  Facility-ID     PIC XXX.
         02  I-S-Info            PIC S9(9) BINARY.
     PROCEDURE DIVISION.
     PARA-CBLQCEN.
    *************************************************
    ** Call CEEQCEN to return the start of the     **
    **     century window                          **
    *************************************************
         CALL "CEEQCEN" USING STARTCW, FC.
    *************************************************
    ** CEEQCEN has no non-zero feedback codes to   **
    **     check, so just display result.          **
    *************************************************
         IF CEE000 of FC  THEN
             DISPLAY "The start of the century "
                 "window is: " STARTCW
         ELSE
             DISPLAY "CEEQCEN failed with msg "
                 Msg-No of FC UPON CONSOLE
             STOP RUN
         END-IF.

         GOBACK.
```

3. Following is an example of CEEQCEN called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMQCEN                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEEQCEN - query the century window   **/
 /**                                                **/
 /** In this example, CEEQCEN is called to query    **/
 /** The date at which the century window starts.   **/
 /** The century window is the 100-year window      **/
 /** within which Language Environment assumes      **/
 /** all two-digit years lie.                       **/
 /**                                                **/
 /****************************************************/
 PLIQCEN: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL STARTCW REAL FIXED BINARY(31,0);
    DCL 01 FC,                    /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    /* Call CEEQCEN to return the start of the       */
    /*    century window                             */
    CALL CEEQCEN ( STARTCW, FC );

    /* CEEQCEN has no non-zero feedback codes        */
    /* to check, so print result                     */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST ( 'The century window starts '
          || STARTCW || ' years before today.');
          END;
       ELSE  DO;
          DISPLAY( 'CEEQCEN failed with msg '
             || FC.MsgNo );
          STOP;
          END;

    END PLIQCEN;
```

CEEQDTC—Query locale date and time conventions

CEEQDTC, analogous to the C language function `localdtconv()`, queries the date and time conventions from the current locale and sets the components of a structure with values appropriate to the settings for the LC_TIME category. CEEQDTC is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►──CEEQDTC──(──version──,──localdt──,──fc──)─────────────────────────────►◄
```

version
This parameter points to a user supplied VERSION_INFO structure, in which the first four bytes contain the callable service version number.

Requirement: If the parameter does not points to a user supplied VERSION_INFO structure, it must be omitted.If the parameter points to a VERSION_INFO containing version number 1, CEEQDTC returns a DTCONV structure with the new element showing the format of the ISO 8601:2000 standard date. If the parameter is omitted, CEEQDTC returns the DTCONV format used before C99. For information about how to code this parameter, see "Invoking callable services" on page 114.

localdt **(output)**
A pointer to the data structure containing the date and time formatting information from the current, active locale. The fields used to populate the structure come from the LC_TIME category. The LC_TIME category structure fields used to retrieve the date and time values are:

abmon
Abbreviated month names (12 instances of a halfword length-prefixed character string, VSTRING, of 22 bytes)

mon Month names (12 instances of a halfword length-prefixed character string, VSTRING, of 62 bytes)

abday Abbreviated day names (7 instances of a halfword length-prefixed character string, VSTRING, of 22 bytes)

day Day names (7 instances of a halfword length-prefixed character string, VSTRING, of 62 bytes)

d_t_fmt
Date and time format (1 instance of a halfword length-prefixed character string, VSTRING, of 82 bytes)

d_fmt Date format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)

t_fmt Time format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)

am_fmt
AM string (1 instance of a halfword length-prefixed character string, VSTRING, of 22 bytes)

pm_fmt
PM string (1 instance of a halfword length-prefixed character string, VSTRING, of 22 bytes)

t_fmt_ampm
Time format ampm (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)

ISO_STD8601_2000
ISO8601:2000 standard date format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.
CEE3VN	3	4087	Input Error: The `version_info` control block contains a version number that is not valid.

Usage notes

- PL/I MTF consideration—CEEQDTC is not supported in PL/I MTF applications.
- If no call to CEESETL has been made, the default locale values to be used are determined at installation time for Language Environment.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- See “CEESETL—Set locale operating environment” on page 411 for more information about the current locale.

Examples

1. Following is an example of CEEQDTC called by COBOL.

```
 CBL LIB,QUOTE
*Module/File Name: IGZTQDTC
*************************************************
*  Example for callable service CEEQDTC         *
*  MAINQDTC - Retrieve date and time convention  *
*             structures for two countries and   *
*             compare an item.                   *
*  Valid only for COBOL for MVS & VM Release 2   *
*  or later.                                     *
```

```
**************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  MAINQDTC.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
* Use DTCONV structure for CEEQDTC calls
 COPY CEEIGZDT.
*
 PROCEDURE DIVISION.
*  Subroutine needed for addressing
     CALL "COBQDTC" USING DTCONV.

     STOP RUN.
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBQDTC.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  Locale-Name.
     02  LN-Length  PIC S9(4) BINARY.
     02  LN-String  PIC X(256).
* Use Locale category constants
 COPY CEEIGZLC.
*
 01  Test-Length1 PIC S9(4) BINARY.
 01  Test-String1 PIC X(80).
 01  Test-Length2 PIC S9(4) BINARY.
 01  Test-String2 PIC X(80).
 01  FC.
     02  Condition-Token-Value.
     COPY CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity  PIC S9(4) BINARY.
             04  Msg-No    PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl  PIC X.
         03  Facility-ID   PIC XXX.
     02  I-S-Info  PIC S9(9) BINARY.
*
 LINKAGE SECTION.
* Use Locale structure DTCONV for CEEQDTC calls
 COPY CEEIGZDT.
*
 PROCEDURE DIVISION USING DTCONV.
*  Set up locale for France
     MOVE 4 TO LN-Length.
     MOVE "FFEY" TO LN-String (1:LN-Length).
*  Call CEESETL to set all locale categories
     CALL "CEESETL" USING Locale-Name, LC-ALL,
                          FC.
*  Check feedback code
     IF Severity > 0
        DISPLAY "Call to CEESETL failed. " Msg-No
        EXIT PROGRAM
     END-IF.

*  Call CEEQDTC for French values
     CALL "CEEQDTC" USING OMITTED,
                          ADDRESS OF DTCONV, FC.
*  Check feedback code
     IF Severity > 0
        DISPLAY "Call to CEEQDTC failed. " Msg-No
        EXIT PROGRAM
```

```
               END-IF.

          *  Save date and time format for FFEY locale
               MOVE D-T-FMT-Length IN DTCONV TO Test-Length1
               MOVE D-T-FMT-String IN DTCONV TO Test-String1

          *  Set up locale for French Canadian
               MOVE 4 TO LN-Length.
               MOVE "FCEY" TO LN-String (1:LN-Length).

          *  Call CEESETL to set locale for all categories
               CALL "CEESETL" USING Locale-Name, LC-ALL,
                                    FC.
          *  Check feedback code
               IF Severity > 0
                  DISPLAY "Call to CEESETL failed. " Msg-No
                  EXIT PROGRAM
               END-IF.

          *  Call CEEQDTC again for French Canadian values
               CALL "CEEQDTC" USING OMITTED,
                                    ADDRESS OF DTCONV, FC.

          *  Check feedback code and display results
               IF Severity = 0
          *  Save date and time format for FCEY locale
                  MOVE D-T-FMT-Length IN DTCONV
                                    TO Test-Length2
                  MOVE D-T-FMT-String IN DTCONV
                                    TO Test-String2
                  IF Test-String1(1:Test-Length1) =
                     Test-String2(1:Test-Length2)
                     DISPLAY "Same date and time format."
                  ELSE
                     DISPLAY "Different formats."
                     DISPLAY  Test-String1(1:Test-Length1)
                     DISPLAY  Test-String2(1:Test-Length2)
                  END-IF
               ELSE
                  DISPLAY "Call to CEEQDTC failed. " Msg-No
               END-IF.

               EXIT PROGRAM.
           END PROGRAM COBQDTC.
          *
           END PROGRAM MAINQDTC.
```

2. Following is an example of CEEQDTC called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMQDTC                          */
 /****************************************************/
 /* Example for callable service CEEQDTC              */
 /* Function: Retrieve date and time convention       */
 /*  structures for two countries, compare an item.  */
 /****************************************************/

 PLIQDTC: PROC OPTIONS(MAIN);

 %INCLUDE CEEIBMAW; /* ENTRY defs, macro defs         */
 %INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants    */
 %INCLUDE CEEIBMLC; /* Locale category constants      */
 %INCLUDE CEEIBMDT; /* DTCONV for CEEQDTC calls       */

 /* use explicit pointer to local DTCONV structure */
 DCL LOCALDT POINTER INIT(ADDR(DTCONV));

 /* CEESETL service call arguments */
```

```
DCL LOCALE_NAME CHAR(256) VARYING;

DCL 1 DTCONVC LIKE DTCONV; /* Def Second Structure  */

DCL 1 FC,                         /* Feedback token */
       3 MsgSev    REAL FIXED BINARY(15,0),
       3 MsgNo     REAL FIXED BINARY(15,0),
       3 Flags,
          5 Case      BIT(2),
          5 Severity  BIT(3),
          5 Control   BIT(3),
       3 FacID     CHAR(3),       /* Facility ID */
       3 ISI       /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);

  /* set locale with IBM default for France         */
  LOCALE_NAME = 'FFEY';   /* or Fr_FR.IBM-1047      */

  /* use LC_ALL category constant from CEEIBMLC */
  CALL CEESETL ( LOCALE_NAME, LC_ALL,  FC );

  /* retrieve date and time structure, France Locale*/
  CALL CEEQDTC ( *, LOCALDT, FC );
  /* set locale with French Canadian(FCEY) defaults */
  /* literal constant -1 used to set all categories */
  CALL CEESETL ( 'FCEY', -1,  FC );

  /* retrieve date and time tables for French Canada*/
  /* example of temp pointer used for service call  */
  CALL CEEQDTC ( *, ADDR(DTCONVC), FC );

  /* compare date and time formats for two countries*/
  IF DTCONVC.D_T_FMT = DTCONV.D_T_FMT THEN
    DO;
      PUT SKIP LIST('Countries have same D_T_FMT' );
    END;
  ELSE
    DO;
      PUT SKIP LIST('Date and Time Format ',
                    DTCONVC.D_T_FMT||' vs '||
                    DTCONV.D_T_FMT );
    END;

END PLIQDTC;
```

CEEQRYL—Query active locale environment

CEEQRYL, analogous to the C language function `localename=setlocale(`*category*`, NULL)`, queries the environment for which locale defines the current setting for the locale category. CEEQRYL is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►──CEEQRYL──(──category──,──localename──,──fc──)──────────────────────────►◄
```

category **(input)**

A symbolic integer number that represents all or part of the locale for a program. Depending on the value of the *localename*, these categories can be initiated by the values of global categories of corresponding names. The following values for the *category* parameter are valid; Language Environment locale callable services do not support the LC_TOD and LC_SYNTAX categories.

LC_ALL
A 4-byte integer (with a value of -1) that affects all locale categories associated with a program's locale.

LC_COLLATE
A 4-byte integer (with a value of 0) that affects the behavior of regular expression and collation subroutines.

LC_CTYPE
A 4-byte integer (with a value of 1) that affects the behavior of regular expression, character classification, case conversion, and wide character subroutines.

LC_MESSAGES
A 4-byte integer (with a value of 6) that affects the format and values for positive and negative responses.

LC_MONETARY
A 4-byte integer (with a value of 3) that affects the behavior of subroutines that format monetary values.

LC_NUMERIC
A 4-byte integer (with a value of 2) that affects the behavior of subroutines that format non-monetary numeric values.

LC_TIME
A 4-byte integer (with a value of 4) that affects the behavior of time conversion subroutines.

localename **(output)**
Returns the name of the locale that describes the current setting of the category requested.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2KD	3	2701	An invalid category parameter was passed to a locale function.
CEE3T1	3	4001	General Failure: Service could not be completed.

Usage notes

- PL/I MTF consideration—CEEQRYL is not supported in PL/I MTF applications.
- CEEQRYL does not change the status of the active locales.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

- If the active locale is not explicitly set with CEESETL or `setlocale(`*`category, localename`*`)`, then the locale chosen is as follows:
  - With POSIX(OFF), the `SAA C` locale is chosen, and querying the locale with CEEQRYL returns "C" as the locale name.
  - With POSIX(ON), the `POSIX C` locale is chosen, and querying the locale with CEEQRYL returns "POSIX" as the locale name.

  The `SAA C` locale provides compatibility with previous releases of C/370. The `POSIX C` locale provides consistency with POSIX requirements and supports the z/OS UNIX System Services environment.

For more information

- See "CEESETL—Set locale operating environment" on page 411 for more information about the current locale.
- For more information about LC_TIME, see "CEEQDTC—Query locale date and time conventions" on page 385.
- For information about the definition of the `SAA C` and `POSIX C` locales and the differences between them, see *z/OS XL C/C++ Programming Guide*.

Examples

For examples of how to use CEEQRYL with other Language Environment locale callable services, see "CEESETL—Set locale operating environment" on page 411.

CEERAN0—Calculate uniform random numbers

CEERAN0 generates a sequence of uniform pseudo-random numbers between 0.0 and 1.0 using the multiplicative congruential method with a user-specified seed. The uniform (0,1) pseudo-random numbers are generated using the multiplicative congruential method:

```
seed(i) = (950706376 * seed(i-1)) mod 2147483647;

randomno(i) = seed(i) / 2147483647;
```

Syntax

```
►►──CEERAN0──(──seed──,──random_no──,──fc──)─────────────────────────────►◄
```

`seed` **`(input/output)`**
: A fullword binary signed integer representing an initial value used to generate random numbers. *seed* must be a variable; it cannot be an input-only parameter. The valid range is 0 to +2,147,483,646. If *seed* equals 0, the seed is generated from the current Greenwich Mean Time. On return to the calling routine, CEERAN0 changes the value of *seed* so that it can be used as the new seed in the next call.

`random_no` **`(output)`**
: An 8-byte double precision floating-point number with a value between 0 and 1, exclusive. If *seed* is invalid, *random_no* is set to -1 and CEERAN0 terminates with a non-CEE000 symbolic feedback code.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2ER	1	2523	The system time was not available when CEERAN0 was called. A seed value of 1 was used to generate a random number and a new seed value.
CEE2ES	3	2524	An invalid seed value was passed to CEERAN0. The random number was set to -1.

Usage notes

- z/OS UNIX considerations—In multithread applications, CEERAN0 affects only the calling thread. The seed is unique to the thread.

Examples

1. Following is an example of CEERAN0 called by C/C++.

```
/*Module/File Name: EDCRAN0   */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main (void) {

   _INT4 seed;
   _FLOAT8 random;
   _FEEDBACK fc;
   int number;

   seed = 0;
   CEERAN0(&seed,&random,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEERAN0 failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   number = random * 1000;
   printf("The 3 digit random number is %d\n",number);
}
```

2. Following is an example of CEERAN0 called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTRAN0
      ************************************************
      ** CBLRAN0 - Call CEERAN0 to generate uniform **
      **                    random numbers         **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLRAN0.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  SEED                    PIC S9(9) BINARY.
       01  RANDNUM                 COMP-2.
```

```
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.

 PROCEDURE DIVISION.

 PARA-CBLRAN0.
************************************************
** Specify 0 for SEED, so the seed will be
** derived from the current Greenwich Mean Time
************************************************
     MOVE 0 TO SEED.
************************************************
** Call CEERAN0 to return random number between
**     0.0 and 1.0
************************************************
     CALL "CEERAN0" USING SEED , RANDNUM , FC.
************************************************
** If CEERAN0 runs successfully, display result.
************************************************
     IF CEE000 of FC  THEN
         DISPLAY "The random number is: " RANDNUM
     ELSE
         DISPLAY "CEERAN0 failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
     GOBACK.
```

3. Following is an example of CEERAN0 called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMRAN0                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEERAN0 - calculate uniform random   **/
 /** numbers                                        **/
 /**                                                **/
 /****************************************************/
 PLIRAN0: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL SEED     REAL FIXED BINARY(31,0);
    DCL RANDNUM  REAL FLOAT DECIMAL(16);
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    SEED = 7; /* Specify an integer as the initial   */
```

```
                /* value used to calculate the random  */
                /* numbers                             */


     /* Call CEERAN0 to generate random number between */
     /*    0.0 and 1.0                                 */
      CALL CEERAN0 ( SEED, RANDNUM, FC );

      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST ( 'The random number is '
            || RANDNUM );
         END;
      ELSE  DO;
         DISPLAY( 'CEERAN0 failed with msg '
            || FC.MsgNo );
         STOP;
         END;

   END PLIRAN0;
```

CEESCEN—Set the century window

CEESCEN sets the century in which Language Environment contains the 2-digit year value. Use it in conjunction with CEEDAYS or CEESECS when:

- You process date values containing 2-digit years (for example, in the YYMMDD format).
- The Language Environment default century interval does not meet the requirements of a particular application.

Century intervals are kept as thread-level data, so changing the interval in one thread does not affect the interval in another thread. To query the century window, use CEEQCEN.

CEEQCEN affects and is affected by only the Language Environment NLS and date and time services, not the Language Environment locale callable services or the C locale-sensitive functions.

Syntax

```
▶▶──CEERAN0──(──century_start──,──fc──)──────────────────────────────────▶◀
```

century_start
: An integer between 0 and 100, setting the century window. A value of 80, for example, places all two-digit years within the 100-year window starting 80 years before the system date. In 1995, therefore, all two-digit years are assumed to represent dates between 1915 and 2014, inclusive.

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E6	3	2502	The UTC/GMT was not available from the system.

Usage notes

- z/OS UNIX considerations—CEESCEN applies to the enclave. The century window applies to the enclave.

For more information

- See "CEEDAYS—Convert date to Lilian format" on page 241 for more information about the CEEDAYS callable service.
- See "CEESECS—Convert timestamp to seconds" on page 405 for more information about the CEESECS callable service.
- See "CEEQCEN—Query the century window" on page 382 for more information about the CEEQCEN callable service.

Examples

1. Following is an example of CEESCEN called by C/C++.

```
/*Module/File Name: EDCSCEN   */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main (void) {

  _INT4 century_start;
  _FEEDBACK fc;

  century_start = 50;

  CEESCEN(&century_start,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEESCEN failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
}
```

2. Following is an example of CEESCEN called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTSCEN
      *************************************************
      **                                             **
      ** CBLSCEN - Call CEESCEN to set the Lang. Env. **
      **           century window                    **
      **                                             **
      ** In this example, CEESCEN is called to change **
      ** the start of the century window to 30 years  **
      ** before the system date. CEEQCEN is then      **
      ** called to query that the change made.  A     **
      ** message that this has been done is then      **
      ** displayed.                                   **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSCEN.
```

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  STARTCW                 PIC S9(9) BINARY.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION.
 PARA-CBLSCEN.
**************************************************
** Specify 30 as century start, and two-digit
**     years will be assumed to lie in the
**     100-year window starting 30 years before
**     the system date.
**************************************************
     MOVE 30 TO STARTCW.

**************************************************
** Call CEESCEN to change the start of the century
**     window.
**************************************************

     CALL "CEESCEN" USING STARTCW, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEESCEN failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
PARA-CBLQCEN.
**************************************************
** Call CEEQCEN to return the start of the century
**     window
**************************************************
     CALL "CEEQCEN" USING STARTCW, FC.

**************************************************
** CEEQCEN has no non-zero feedback codes to
**     check, so just display result.
**************************************************
         DISPLAY "The start of the century "
             "window is: " STARTCW
     GOBACK.
```

3. Following is an example of CEESCEN called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMSCEN                        */
 /****************************************************/
 /**                                                **/
 /** Function: CEESCEN - set the century window     **/
 /**                                                **/
 /****************************************************/
 PLISCEN: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL STARTCW  REAL FIXED BINARY(31,0);
    DCL 01 FC,                      /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    STARTCW = 20; /* Set 20 as century start */

   /* Call CEESCEN to request that two-digit years   */
   /*    lie in the 100-year window starting 20      */
   /*    years before the system date                */
    CALL CEESCEN ( STARTCW, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST ( 'The century window now starts '
          || STARTCW || ' years before today.');
       END;
    ELSE  DO;
       DISPLAY( 'CEESCEN failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLISCEN;
```

CEESCOL—Compare collation weight of two strings

CEESCOL, which is analogous to the C language function `strcoll()`, compares two character strings based on the collating sequence specified in the LC_COLLATE category of the current locale. CEESCOL associates a collation weight with every character in the code set for the locale. The characters in the input strings are converted to their collation weights and then compared on the basis of these weights. CEESCOL is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►──CEESCOL──(──omitted_parm──,──string1──,──string2──,──result──,──fc──)──────►◄
```

omitted_parm
: This parameter is reserved for future expansion and must be omitted. For information about how to code an omitted parameter, see "Invoking callable services" on page 114.

string1 **(input)**
: A halfword length-prefixed character string (VSTRING) with a maximum length of 4K bytes. *string1* points to a string of characters that are to be compared against *string2* .

string2 **(input)**
: A halfword length-prefixed character string (VSTRING) with a maximum length of 4K bytes. *string2* points to a string of characters that *string1* is compared against.

result **(output)**
Specifies the result of the string comparison. If successful, the following values are returned:
- Less than 0, if *string1* is less than *string2*
- Equal to 0, if *string1* is equal to *string2*
- Greater than 0, if *string1* is greater than *string2*

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.

Usage notes

- PL/I MTF consideration—CEESCOL is not supported in PL/I MTF applications.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- See "CEESETL—Set locale operating environment" on page 411 for more information about the CEESETL callable service.

Examples

1. Following is an example of CEESCOL called by COBOL.

```
 CBL LIB,QUOTE
*Module/File Name: IGZTSCOL
*************************************************
*  Example for callable service CEESCOL         *
*   COBSCOL - Compare two character strings     *
*             and print the result.             *
*  Valid only for COBOL for MVS & VM Release 2  *
*  or later.                                    *
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBSCOL.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  String1.
     02  Str1-Length  PIC S9(4) BINARY.
     02  Str1-String.
         03  Str1-Char  PIC X
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Str1-Length.
 01  String2.
     02  Str2-Length  PIC S9(4) BINARY.
     02  Str2-String.
         03  Str2-Char  PIC X
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Str2-Length.
 01  Result  PIC S9(9)  BINARY.
 01  FC.
     02  Condition-Token-Value.
```

```
         COPY CEEIGZCT.
             03  Case-1-Condition-ID.
                 04  Severity    PIC S9(4) BINARY.
                 04  Msg-No      PIC S9(4) BINARY.
             03  Case-2-Condition-ID
                       REDEFINES Case-1-Condition-ID.
                 04  Class-Code  PIC S9(4) BINARY.
                 04  Cause-Code  PIC S9(4) BINARY.
             03  Case-Sev-Ctl    PIC X.
             03  Facility-ID     PIC XXX.
         02  I-S-Info            PIC S9(9) BINARY.
    *
     PROCEDURE DIVISION.
    **************************************************
    *  Set up two strings for comparison
    **************************************************
         MOVE 9 TO Str1-Length.
         MOVE "12345a789"
            TO Str1-String (1:Str1-Length)
         MOVE 9 TO Str2-Length.
         MOVE "12346$789"
            TO Str2-String (1:Str2-Length)
    **************************************************
    *  Call CEESCOL to compare the strings
    **************************************************
         CALL "CEESCOL" USING OMITTED, String1,
                              String2, Result, FC.

    **************************************************
    *  Check feedback code
    **************************************************
         IF Severity > 0
            DISPLAY "Call to CEESCOL failed. " Msg-No
            STOP RUN
         END-IF.

    **************************************************
    *  Check result of compare
    **************************************************
         EVALUATE TRUE
            WHEN Result < 0
               DISPLAY "1st string < 2nd string."
            WHEN Result > 0
               DISPLAY "1st string > 2nd string."
            WHEN OTHER
               DISPLAY "Strings are identical."
         END-EVALUATE.

         STOP RUN.
     END PROGRAM COBSCOL.
```

2. Following is an example of CEESCOL called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMSCOL                        */
 /***************************************************/
 /* Example for callable service CEESCOL            */
 /* Function: Compare two character strings and     */
 /*  print the result.                              */
 /***************************************************/

 PLISCOL: PROC OPTIONS(MAIN);

 %INCLUDE CEEIBMAW; /* ENTRY defs, macro defs for Language Environment */
 %INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants                     */
 %INCLUDE CEEIBMLC; /* Locale category constants                       */

 /* CEESCOL service call arguments */
```

```
DCL STRING1 CHAR(256) VARYING;/* first string       */
DCL STRING2 CHAR(256) VARYING;/* second string      */
DCL RESULT_SCOL BIN FIXED(31);/* result of compare  */

DCL 01 FC,                        /* Feedback token */
       03 MsgSev    REAL FIXED BINARY(15,0),
       03 MsgNo     REAL FIXED BINARY(15,0),
       03 Flags,
          05 Case      BIT(2),
          05 Severity  BIT(3),
          05 Control   BIT(3),
       03 FacID     CHAR(3),         /* Facility ID */
       03 ISI      /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);

  STRING1 = '12345a789';
  STRING2 = '12346$789';

  CALL CEESCOL( *, STRING1, STRING2, RESULT_SCOL,FC);

  /* FBCHECK macor used (defined in CEEIBMCT) */
  IF FBCHECK( FC, CEE3T1 ) THEN
    DO;
      DISPLAY ('CEESCOL not completed '||FC.MsgNo );
      STOP;
    END;

  SELECT;
  WHEN( RESULT_SCOL < 0 )
    PUT SKIP LIST(
      '"firststring" is less than "secondstring" ');
  WHEN( RESULT_SCOL > 0 )
    PUT SKIP LIST(
    '"firststring" is greater than "secondstring" ');
  OTHERWISE
    PUT SKIP LIST( 'Strings are identical' );
  END;  /* END SELECT */

END PLISCOL;
```

CEESECI—Convert seconds to integers

CEESECI converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. Use CEESECI instead of CEEDATM when the output is needed in numeric format rather than in character format.

The inverse of CEESECI is CEEISEC, which converts integer year, month, day, hour, second, and millisecond to number of seconds. If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86,400 (number of seconds in a day), and pass the new value to CEESECI.

Syntax

```
►►──CEESECI──(──input_seconds──,──output_year──,──output_month──,──────────────►

►──output_day──,──output_hours──,──output_minutes──,──output_seconds──,──────────►

►──output_milliseconds──,──fc──)──────────────────────────────────────────────►◄
```

input_seconds
: A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range for *input_seconds* is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999). If *input_seconds* is invalid, all output parameters except the feedback code are set to 0.

output_year **(output)**
: A 32-bit binary integer representing the year. The range of valid *output_year*s is 1582 to 9999, inclusive.

output_month **(output)**
: A 32-bit binary integer representing the month. The range of valid *output_month*s is 1 to 12.

output_day **(output)**
: A 32-bit binary integer representing the day. The range of valid *output_day*s is 1 to 31.

output_hours **(output)**
: A 32-bit binary integer representing the hour. The range of valid *output_hours* is 0 to 23.

output_minutes **(output)**
: A 32-bit binary integer representing the minutes. The range of valid *output_minutes* is 0 to 59.

output_seconds **(output)**
: A 32-bit binary integer representing the seconds. The range of valid *output_seconds* is 0 to 59.

output_milliseconds **(output)**
: A 32-bit binary integer representing milliseconds. The range of valid *output_milliseconds* is 0 to 999.

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2E9	3	2505	The input_seconds value in a call to CEEDATM or CEESECI was not within the supported range.

Usage notes

- z/OS UNIX consideration—In multithread applications, CEESECI affects only the calling thread.

For more information

- For more information about the CEEISEC callable service, see "CEEISEC—Convert integers to seconds" on page 330.

Examples

1. Following is an example of CEESECI called by C/C++.

```
/*Module/File Name: EDCSECI   */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _INT4 year, month, day, hours, minutes, seconds,
        millisecs;
  _FLOAT8 input;
  _FEEDBACK fc;

  input = 13166064000.0;
  CEESECI(&input,&year,&month,&day,&hours,&minutes,;
          &seconds,&millisecs,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEESECI failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  printf("%f seconds corresponds to the date"
         " %d:%d:%d.%d %d/%d/%d\n",input,hours,minutes,
          seconds,millisecs,month,day,year);
}
```

2. Following is an example of CEESECI called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTSECI
      *************************************************
      **                                             **
      ** CBLSECI - Call CEESECI to convert seconds   **
      **           to integers                       **
      **                                             **
      ** In this example a call is made to CEESECI   **
      ** to convert a number representing the number **
      ** of seconds since 00:00:00 14 October 1582   **
      ** to seven binary integers representing year, **
      ** month, day, hour, minute, second, and       **
      ** millisecond.  The results are displayed in  **
      ** this example.                               **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSECI.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  INSECS                 COMP-2.
       01  YEAR                   PIC S9(9) BINARY.
```

```
 01  MONTH                  PIC S9(9) BINARY.
 01  DAYS                   PIC S9(9) BINARY.
 01  HOURS                  PIC S9(9) BINARY.
 01  MINUTES                PIC S9(9) BINARY.
 01  SECONDS                PIC S9(9) BINARY.
 01  MILLSEC                PIC S9(9) BINARY.
 01  IN-DATE.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char        PIC X,
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of IN-DATE.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char        PIC X,
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                       of PICSTR.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info           PIC S9(9) BINARY.
 PROCEDURE DIVISION.
 PARA-CBLSECS.
*************************************************
** Call CEESECS to convert timestamp of 6/2/88
**     at 10:23:45 AM to Lilian representation
*************************************************
     MOVE 20 TO Vstring-length of IN-DATE.
     MOVE "06/02/88 10:23:45 AM"
             TO Vstring-text of IN-DATE.
     MOVE 20 TO Vstring-length of PICSTR.
     MOVE "MM/DD/YY HH:MI:SS AM"
             TO Vstring-text of PICSTR.
     CALL "CEESECS" USING IN-DATE, PICSTR,
                          INSECS, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEESECS failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
 PARA-CBLSECI.
*************************************************
** Call CEESECI to convert seconds to integers
*************************************************
     CALL "CEESECI" USING INSECS, YEAR, MONTH,
                          DAYS, HOURS,  MINUTES,
                          SECONDS, MILLSEC, FC.
*************************************************
** If CEESECI runs successfully, display results
*************************************************
     IF CEE000 of FC  THEN
         DISPLAY "Input seconds of " INSECS
             " represents:"
         DISPLAY "   Year......... " YEAR
         DISPLAY "   Month........ " MONTH
```

```
            DISPLAY "   Day.......... " DAYS
            DISPLAY "   Hour......... " HOURS
            DISPLAY "   Minute....... " MINUTES
            DISPLAY "   Second....... " SECONDS
            DISPLAY "   Millisecond.. " MILLSEC
        ELSE
            DISPLAY "CEESECI failed with msg "
                Msg-No of FC UPON CONSOLE
            STOP RUN
        END-IF.

        GOBACK.
```

3. Following is an example of CEESECI called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMSECI                          */
 /****************************************************/
 /**                                                  */
 /** Function: CEESECI - convert seconds to integers */
 /**                                                  */
 /****************************************************/
 PLISECI: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL INSECS   REAL FLOAT DECIMAL(16);
    DCL YEAR     REAL FIXED BINARY(31,0);
    DCL MONTH    REAL FIXED BINARY(31,0);
    DCL DAYS     REAL FIXED BINARY(31,0);
    DCL HOURS    REAL FIXED BINARY(31,0);
    DCL MINUTES  REAL FIXED BINARY(31,0);
    DCL SECONDS  REAL FIXED BINARY(31,0);
    DCL MILLSEC  REAL FIXED BINARY(31,0);
    DCL 01 FC,                      /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    INSECS = 13166064060; /* number of seconds since */
                          /* 14 October 1582         */

    /* Call CEESECI to convert INSECS to separate    */
    /*    binary integers representing the year,     */
    /*    month, day, hours, minutes, seconds and    */
    /*    milliseconds.                              */
    CALL CEESECI ( INSECS, YEAR, MONTH, DAYS,
       HOURS, MINUTES, SECONDS, MILLSEC, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT EDIT( INSECS, ' seconds corresponds to ',
           MONTH, '/', DAYS, '/', YEAR, ' at ', HOURS,
           ':', MINUTES, ':', SECONDS, '.', MILLSEC )
           (SKIP, F(9), A, 2 (P'99',A), P'9999', A,
           3 (P'99', A), P'999' );
       END;
    ELSE  DO;
       DISPLAY( 'CEESECI failed with msg '
          || FC.MsgNo );
          STOP;
          END;

      END PLISECI;
```

CEESECS—Convert timestamp to seconds

CEESECS converts a string representing a timestamp into the number of Lilian seconds (number of seconds since 00:00:00 14 October 1582). This service makes it easier to perform time arithmetic, such as calculating the elapsed time between two timestamps.

CEESECS is affected only by the country code setting of the COUNTRY runtime option or CEE3CTY callable service, not the CEESETL callable service or the `setlocale()` function.

The inverse of CEESECS is CEEDATM, which converts *output_seconds* to character format. By default, 2-digit years lie within the 100 year range starting 80 years prior to the system date. Thus, in 1995, all 2-digit years represent dates between 1915 and 2014, inclusive. You can change this range by using the callable service CEESCEN.

Syntax

```
►►—CEESECS—(—input_timestamp—,—picture_string—,—output_seconds—,———————►
►—fc—)————————————————————————————————————————————————————————————————►◄
```

`input_timestamp` **(input)**

A length-prefixed character string representing a date or timestamp in a format matching that specified by *picture_string*. The character string must contain between 5 and 80 picture characters, inclusive. *input_timestamp* can contain leading or trailing blanks. Parsing begins with the first nonblank character (unless the picture string itself contains leading blanks; in this case, CEESECS skips exactly that many positions before parsing begins).

After a valid date is parsed, as determined by the format of the date you specify in *picture_string*, all remaining characters are ignored by CEESECS. Valid dates range between and including the dates 15 October 1582 to 31 December 9999. A full date must be specified. Valid times range from 00:00:00.000 to 23:59:59.999.

As the following example shows, if any part or all of the time value is omitted, zeros are substituted for the remaining values.

```
1992-05-17-19:02 is equivalent to 1992-05-17-19:02:00
1992-05-17       is equivalent to 1992-05-17-00:00:00
```

`picture_string` **(input)**

A halfword length-prefixed character string (VSTRING), indicating the format of the date or timestamp value specified in *input_timestamp*. Each character in the *picture_string* represents a character in *input_timestamp*. For example, if you specify MMDDYY HH.MI.SS as the *picture_string*, CEESECS reads an *input_char_date* of 060288 15.35.02 as 3:35:02 PM on 02 June 1988. If delimiters such as the slash (/) appear in the picture string, leading zeros can be omitted. For example, the following calls to CEESECS all assign the same value to variable *secs*:

```
CALL CEESECS('92/06/03 15.35.03', 'YY/MM/DD
             HH.MI.SS', secs, fc);
CALL CEESECS('92/6/3 15.35.03'  , 'YY/MM/DD
             HH.MI.SS', secs, fc);
CALL CEESECS('92/6/3 3.35.03 PM', 'YY/MM/DD
             HH.MI.SS AP', secs, fc);
CALL CEESECS('92.155 3.35.03 pm', 'YY.DDD
             HH.MI.SS AP', secs, fc);
```

If picture string is left null or blank, CEESECS gets *picture_string* based on the current value of the COUNTRY runtime option. For example, if the current value of the COUNTRY runtime option is FR (France), the date format would be DD.MM.YYYY.

If *picture_string* includes a Japanese era symbol `<JJJJ>`, the `YY` position in *input_timestamp* represents the year number within the Japanese era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 35 on page 488 for a list of Japanese eras supported by CEESECS.

If *picture_string* includes era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *input_timestamp* represents the year number within the era.

See Table 33 on page 487 for a list of valid picture characters, and Table 34 on page 488 for examples of valid picture strings.

output_seconds **(output)**

A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second 86,401 (24*60*60 + 01) in the Lilian format. 19:00:01.12 on 16 May 1988 is second 12,799,191,601.12.

The largest value represented is 23:59:59.999 on 31 December 9999, which is second 265,621,679,999.999 in the Lilian format.

A 64-bit double floating-point value can accurately represent approximately 16 significant decimal digits without loss of precision. Therefore, accuracy is available to the nearest millisecond (15 decimal digits).

If *input_timestamp* does not contain a valid date or timestamp, *output_seconds* is set to 0 and CEESECS terminates with a non-CEE000 symbolic feedback code.

Elapsed time calculations are performed easily on the *output_seconds*, because it represents elapsed time. Leap year and end-of-year anomalies do not affect the calculations.

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Code	Severity	Message number	Message text
CEE2EB	3	2507	Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.
CEE2EC	3	2508	The date value passed to CEEDAYS or CEESECS was invalid.
CEE2ED	3	2509	The era passed to CEEDAYS or CEESECS was not recognized.
CEE2EE	3	2510	The hours value in a call to CEEISEC or CEESECS was not recognized.
CEE2EH	3	2513	The input date passed in a CEEISEC, CEEDAYS, or CEESECS call was not within the supported range.
CEE2EK	3	2516	The minutes value in a CEEISEC call was not recognized.
CEE2EL	3	2517	The month value in a CEEISEC call was not recognized.
CEE2EM	3	2518	An invalid picture string was specified in a call to a date/time service.
CEE2EN	3	2519	The seconds value in a CEEISEC call was not recognized.
CEE2EP	3	2521	The year-within-era value passed to CEEDAYS or CEESECS was zero.
CEE2ET	3	2525	CEESECS detected non-numeric data in a numeric field, or the timestamp string did not match the picture string.

Usage notes

- The probable cause for receiving message number 2518 is a picture string that contains an invalid DBCS string. You should verify that the data in the picture string is correct.
- z/OS UNIX consideration—In multithread applications, CEESECS affects only the calling thread.

For more information

- See “CEESCEN—Set the century window” on page 394 for more information about the CEESCEN callable service.
- See “COUNTRY” on page 22 for more information about the COUNTRY runtime option.

Examples

1. Following is an example of CEESECS called by C/C++.

```
/*Module/File Name: EDCSECS   */

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

   _FEEDBACK fc;
   _FLOAT8 seconds1, seconds2;
   _VSTRING date,date_pic;

   /* use CEESECS to convert to seconds timestamp */
   strcpy(date.string,"09/13/91 23:23:23");
   date.length = strlen(date.string);
   strcpy(date_pic.string,"MM/DD/YY HH:MI:SS");
   date_pic.length = strlen(date_pic.string);

   CEESECS(&date,&date_pic,&seconds1,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEESECS failed with message number %d\n",
           fc.tok_msgno);
      exit(2999);
   }

   strcpy(date.string,
            "December 15, 1992 at 8:23:45 AM");
   date.length = strlen(date.string);
   strcpy(date_pic.string,
              "Mmmmmmmmmmmmz DD, YYYY at ZH:MI:SS AP");
   date_pic.length = strlen(date_pic.string);

   CEESECS(&date,&date_pic,&seconds2,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEESECS failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }

   printf("The number of seconds between:\n");
   printf(" September 13, 1991 at 11:23:23 PM");
   printf(
    " and December 15, 1992 at 8:23:45 AM is:\n  %f\n",
                 seconds2 - seconds1);
}
```

2. Following is an example of CEESECS called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTSECS
      *************************************************
      ** CBLSECS - Call CEESECS to convert           **
      **           timestamp to number of seconds    **
      **                                             **
      ** In this example, calls are made to CEESECS **
      ** to convert two timestamps to the number of **
      ** seconds since 00:00:00 14 October 1582.    **
      ** The Lilian seconds for the earlier         **
      ** timestamp are then subtracted from the     **
      ** Lilian seconds for the later timestamp     **
      ** to determine the number of between the     **
      ** two.  This result is displayed.            **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSECS.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  SECOND1                 COMP-2.
       01  SECOND2                 COMP-2.
       01  TIMESTP.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                              of TIMESTP.
       01  TIMESTP2.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
```

```
                       of TIMESTP2.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char        PIC X,
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of PICSTR.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
 PROCEDURE DIVISION.
 PARA-SECS1.

************************************************
** Specify first timestamp and a picture string
**     describing the format of the timestamp
**     as input to CEESECS
************************************************
     MOVE 25 TO Vstring-length of TIMESTP.
     MOVE "1969-05-07 12:01:00.000"
             TO Vstring-text of TIMESTP.
     MOVE 25 TO Vstring-length of PICSTR.
     MOVE "YYYY-MM-DD HH:MI:SS.999"
             TO Vstring-text of PICSTR.
************************************************
** Call CEESECS to convert the first timestamp
** to Lilian seconds
************************************************
     CALL "CEESECS" USING TIMESTP, PICSTR,
                          SECOND1, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEESECS failed with msg "
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.
 PARA-SECS2.
************************************************
** Specify second timestamp and a picture string
**     describing the format of the timestamp as
**     input to CEESECS.
************************************************
     MOVE 25 TO Vstring-length of TIMESTP2.
     MOVE "2000-01-01 00:00:01.000"
             TO Vstring-text of TIMESTP2.
     MOVE 25 TO Vstring-length of PICSTR.
     MOVE "YYYY-MM-DD HH:MI:SS.999"
             TO Vstring-text of PICSTR.

************************************************
** Call CEESECS to convert the second timestamp
**     to Lilian seconds
************************************************
     CALL "CEESECS" USING TIMESTP2, PICSTR,
                          SECOND2, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY "CEESECS failed with msg "
```

```
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

       PARA-SECS2.
      *************************************************
      ** Subtract SECOND2 from SECOND1 to determine the
      **     number of seconds between the two timestamps
      *************************************************
           SUBTRACT SECOND1 FROM SECOND2.
           DISPLAY "The number of seconds between "
               Vstring-text OF TIMESTP " and "
               Vstring-text OF TIMESTP2 " is: " SECOND2.

           GOBACK.
```

3. Following is an example of CEESECS called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMSECS                          */
 /****************************************************/
 /**                                                 */
 /** Function: CEESECS - Change timestamp to seconds */
 /**                                                 */
 /** In this example, CEESECS is called to return an */
 /** input timestamp as the number of seconds since  */
 /** 14 October 1582.                                */
 /**                                                 */
 /****************************************************/
 PLISECS: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL TIMESTP CHAR(255) VARYING;
    DCL PICSTR CHAR(255) VARYING;
    DCL SECONDS REAL FLOAT DECIMAL(16);
    DCL 01 FC,                      /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);

    TIMESTP = '10 November 1992'; /* Specify input   */
                                  /* date as timestamp */
    PICSTR  = 'ZD Mmmmmmmmmmmmmmz YYYY';
           /* Picture string that descibes timestamp */

    /* Call CEESECS to return the input date as      */
    /*    Lilian seconds                             */
    CALL CEESECS ( TIMESTP, PICSTR, SECONDS, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'There were ' || SECONDS
          || ' seconds between 14 Oct 1582 and '
          || TIMESTP );
       END;
    ELSE  DO;
       DISPLAY( 'CEESECS failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLISECS;
```

CEESETL—Set locale operating environment

CEESETL, analogous to the C language function `setlocale()`, establishes a global locale operating environment, which determines the behavior of character collation, character classification, date and time formatting, numeric punctuation, and positive/negative response patterns. CEESETL is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►──CEESETL──(──localename──,──category──,──fc──)──────────────────────────►◄
```

localename **(input)**
A halfword length-prefixed character string (VSTRING) with a maximum of 256 bytes. *localename* is a valid locale name known to the locale model. The category named in the call is set according to the named locale. If *localename* is null or blank, CEESETL sets the locale environment according to the environment variables. This is the equivalent to specifying `setlocale(LC_ALL,"")`. If these environment variables are defined, you can locate them in the following order:

- LC_ALL if it is defined and not null
- LANG if it is defined and not null
- The default C locale

category **(input)**
A symbolic integer number that represents all or part of the locale for a program. Depending on the value of the *localename*, these categories can be initiated by the values of global categories of corresponding names. The following values for the *category* parameter are valid:

LC_ALL
A 4-byte integer (with a value of -1) that affects all locale categories associated with a program's locale.

LC_COLLATE
A 4-byte integer (with a value of 0) that affects the behavior of regular expression and collation subroutines.

LC_CTYPE
A 4-byte integer (with a value of 1) that affects the behavior of regular expression, character classification, case conversion, and wide character subroutines.

LC_MESSAGES
A 4-byte integer (with a value of 6) that affects the format and values for positive and negative responses.

LC_MONETARY
A 4-byte integer (with a value of 3) that affects the behavior of subroutines that format monetary values.

LC_NUMERIC
A 4-byte integer (with a value of 2) that affects the behavior of subroutines that format non-monetary numeric values.

LC_TIME
A 4-byte integer (with a value of 4) that affects the behavior of time conversion subroutines.

Language Environment locale callable services do not support the LC_TOD and LC_SYNTAX categories.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2KD	3	2701	An invalid category parameter was passed to a locale function.
CEE2KE	3	2702	An invalid locale name parameter was passed to a locale function.
CEE3T1	3	4001	General Failure: Service could not be completed.

Usage notes

- PL/I MTF consideration—CEESETL is not supported in PL/I MTF applications.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.
- The LC_ALL category indicates that all categories are to be changed with regard to the specific locale. The LC_ALL value, when set by CEESETL, becomes the current values for all six LC_* categories.
- If the active locale is not explicitly set with CEESETL or `setlocale(`*`category`*`,` *`localename`*`)`, then the locale chosen is as follows:
  - With POSIX(OFF), the `SAA C` locale is chosen, and querying the locale with CEEQRYL returns "C" as the locale name.
  - With POSIX(ON), the `POSIX C` locale is chosen, and querying the locale with CEEQRYL returns "POSIX" as the locale name.

  The `SAA C` locale provides compatibility with previous releases of C/370. The `POSIX C` locale provides consistency with POSIX requirements and supports the z/OS UNIX environment.

For more information

- See *z/OS XL C/C++ Runtime Library Reference* for details of how various locale categories affect C/C++ language functions.
- For more information about specifying environment variables to set the locale, see *z/OS XL C/C++ Programming Guide*.
- For information about the definition of the `SAA C` and `POSIX C` locales and the differences between them, see *z/OS XL C/C++ Programming Guide*.
- For more information about LC_TIME, see "CEEQDTC—Query locale date and time conventions" on page 385.

Examples

1. Following is an example of CEESETL called by COBOL.

```
 CBL LIB,QUOTE
*Module/File Name: IGZTSETL
*************************************************
*  Example for callable service CEESETL        *
*   COBSETL - Set all global locale environment *
*              categories to country Sweden.    *
*             Query one category.               *
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBSETL.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  Locale-Name.
     02  LN-Length  PIC S9(4) BINARY.
     02  LN-String  PIC X(256).
 01  Locale-Time.
     02  LT-Length  PIC S9(4) BINARY.
     02  LT-String  PIC X(256).
* Use Locale category constants
 COPY CEEIGZLC.
*
 01  FC.
     02  Condition-Token-Value.
     COPY CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) BINARY.
             04  Msg-No      PIC S9(4) BINARY.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) BINARY.
             04  Cause-Code  PIC S9(4) BINARY.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) BINARY.
*
 PROCEDURE DIVISION.
*************************************************
*  Set up locale name for Sweden
*************************************************
     MOVE 14 TO LN-Length.
     MOVE 'Sv_SE.IBM-1047'
             TO LN-String (1:LN-Length).

*************************************************
*  Set all locale categories to Sweden
*  Use LC-ALL category constant from CEEIGZLC
*************************************************
     CALL 'CEESETL' USING Locale-Name, LC-ALL,
                          FC.

*************************************************
*  Check feedback code
*************************************************
     IF Severity > 0
        DISPLAY 'Call to CEESETL failed. ' Msg-No
        STOP RUN
     END-IF.
*************************************************
*  Retrieve active locale for LC-TIME category
*************************************************
     CALL 'CEEQRYL' USING LC-TIME, Locale-Time,
                          FC.

*************************************************
*  Check feedback code and correct locale
*************************************************
     IF Severity = 0
```

```
               IF LT-String(1:LT-Length) =
                       LN-String(1:LN-Length)
                  DISPLAY 'Successful query.'
               ELSE
                  DISPLAY 'Unsuccessful query.'
               END-IF
            ELSE
               DISPLAY 'Call to CEEQRYL failed. ' Msg-No
            END-IF.

            STOP RUN.
        END PROGRAM COBSETL.
```

2. Following is an example of CEESETL called by COBOL.

```
*PROCESS MACRO;
 /*Module/File Name: IBMSETL                         */
 /***************************************************/
 /* Example for callable service CEESETL             */
 /* Function: Set all global locale environment      */
 /*  categories to country. Query one category.      */
 /***************************************************/

 PLISETL: PROC OPTIONS(MAIN);

 %INCLUDE CEEIBMAW; /* ENTRY defs, macro defs        */
 %INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants   */
 %INCLUDE CEEIBMLC; /* Locale category constants     */

 /* CEESETL service call arguments */
 DCL LOCALE_NAME CHAR(14) VARYING;

 /* CEEQRYL service call arguments */
 DCL LOCALE_NAME_TIME CHAR(256) VARYING;

 DCL 01 FC,                        /* Feedback token */
        03 MsgSev    REAL FIXED BINARY(15,0),
        03 MsgNo     REAL FIXED BINARY(15,0),
        03 Flags,
           05 Case      BIT(2),
           05 Severity  BIT(3),
           05 Control   BIT(3),
        03 FacID     CHAR(3),       /* Facility ID */
        03 ISI      /* Instance-Specific Information */
                     REAL FIXED BINARY(31,0);
   /* init locale name with IBM default for Sweden  */
   LOCALE_NAME = 'Sv_SE.IBM-1047';

   /* use LC_ALL category const from CEEIBMLC */
   CALL CEESETL ( LOCALE_NAME, LC_ALL, FC );

   /* FBCHECK macro used (defined in CEEIBMCT) */
   IF FBCHECK( FC, CEE2KE ) THEN
     DO;  /* invalid locale name */
       DISPLAY ( 'Locale LC_ALL Call '||FC.MsgNo );
       STOP;
     END;

   /* retrieve active locale for LC_TIME category */
   /* use LC_TIME category const from CEEIBMLC */
   CALL CEEQRYL ( LC_TIME, LOCALE_NAME_TIME, FC );

   IF FBCHECK( FC, CEE000 ) THEN
     DO;  /* successful query, check category name */
       IF LOCALE_NAME_TIME ¬= LOCALE_NAME THEN
         DO;
           DISPLAY ( 'Invalid LOCALE_NAME_TIME ' );
           STOP;
```

```
          END;
        ELSE
          DO;
            PUT SKIP LIST('Successful query LC_TIME',
                           LOCALE_NAME_TIME);
          END;
      END;
    ELSE
      DO;
        DISPLAY ( 'LC_TIME Category Call '||FC.MsgNo );
        STOP;
      END;

  END PLISETL;
```

CEESGL—Signal a condition

CEESGL raises, or signals, a condition to the Language Environment condition manager. It also provides qualifying data and creates an ISI for a particular instance of the condition. The ISI contains information used by the Language Environment condition manager to identify and react to conditions.

CEESGL is typically used to generate application-specific conditions that are recognized by condition handlers registered using CEEHDLR. Conditions can also be selected to simulate a Language Environment or system condition. If you plan on using a routine that signals a new condition with a call to CEESGL, you should first call the CEECMI callable service to copy any insert information into the ISI associated with the condition.

CEESGL generates a Language Environment condition. You can map some of the Language Environment condition tokens to POSIX signals. Unique conditions signaled by CEESGL are considered to be enabled under Language Environment. Therefore, they undergo Language Environment condition handling.

CEESGL can signal a POSIX condition. If CEESGL signals a POSIX condition and the signal is blocked at the time of the generation but later unblocked and delivered, the POSIX signal processing semantics are applied. The Language Environment synchronous condition manager semantics do not apply.

Severity 0 and 1 conditions are considered safe conditions. They can be ignored if they are not handled and if no feedback token is passed when the condition is raised.

Each signaled condition (of severity 2 or above) increments the error count by one. If the error count exceeds the error count limit (as specified by the ERRCOUNT runtime option—see “ERRCOUNT” on page 31) the condition manager terminates the enclave with abend code 4091, reason code 11. T_I_U (Termination Imminent due to an Unhandled Condition) is not issued. Promoted conditions do not increment the error count. A program established using the CEEHDLR callable service or one of the HLL condition handlers, can then process the raised condition.

ERRCOUNT applies to CEESGL only if the condition generated by CEESGL is delivered synchronously. POSIX signal handling semantics are then applied to the condition.

Table 24 on page 203 contains a list of the S/370 program interrupt codes and their corresponding Language Environment condition token names and message numbers.

Syntax

```
►►──CEESGL──(──cond_rep──,──┬──────────────────┬──fc──)────────────────────►◄
                            └──q_data_token──,──┘
```

cond_rep **(input)**
A 12-byte condition token representing the condition to be signaled. You can either construct your own condition token or use one that Language Environment has already defined. Conditions signaled by CEESGL are not necessarily handled by Language Environment. If you call CEESGL with a *cond_rep*, Language Environment passes control to the language in which the routine is written. The condition manager then determines if it should handle the condition. If so, the HLL handles the condition. If not, control returns to Language Environment. The condition might also be ignored or blocked, or might result in enclave termination.

q_data_token **(input/output)**
An optional 32-bit data object placed in the ISI to access the qualifying data (q_data) associated with the given instance of the condition. The *q_data_token* is a list of information addresses a user condition handler uses to specifically identify and, if necessary, react to, a given condition. The information in the *q_data_token* provides a mechanism by which user-written condition handlers can provide a complete fix-up of some conditions. The *q_data token* associated with a condition using CEESGL can be extracted later using the CEEGQDT callable service.

fc **(output / optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE069	0	0201	An unhandled condition was returned in a feedback code.
CEE0CE	1	0398	Resume with new input.
CEE0CF	1	0399	Resume with new output.
CEE0EB	3	0459	Not enough storage was available to create a new instance specific information block.
CEE0EE	3	0462	Instance specific information for the condition token with message number *message-number* and facility ID *facility-id* could not be found.

Usage notes

- PL/I consideration: Conditions with a facility_ID of IBM cannot be used in CEESGL.

- z/OS UNIX consideration: In multithread applications, CEESGL affects only the calling thread. Delivery of a CEESGL generated condition is limited to the thread that generated the condition. However, if the condition is a severity 2 or higher, and is not handled by the application, the default action of terminate applies to the enclave, not just the calling thread.
- COBOL consideration: When a resume occurs and control resumes to the next instruction that follows the call to CEE3SRP, the COBOL RETURN-CODE special register contains an unpredictable value.

For more information

- See "CEECMI—Store and load message insert data" on page 216 for more information about the CEECMI callable service.
- See *z/OS Language Environment Programming Guide* for more information about mapping Language Environment condition tokens to POSIX signals.
- See "ERRCOUNT" on page 31 for more information about the ERRCOUNT runtime option.
- To construct your own condition token, see "CEENCOD—Construct a condition token" on page 377.
- For more information about condition handling, see *z/OS Language Environment Programming Guide*.
- See "CEEGQDT—Retrieve q_data_token" on page 303 for more information about the CEEGQDT callable service.

Examples

1. Following is an example of CEESGL called by C/C++.

```
/*Module/File Name: EDCSGL    */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;

 /* .
    .
    . */
  /* build the condition token */
  c_1 = 1;
  c_2 = 99;
  cond_case = 1;
  sev = 1;
  control = 0;
  memcpy(facid,"ZZZ",3);
  isi = 0;

  CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
          facid,&isi,&condtok,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
```

```
      exit(2999);
   }

   /* signal the condition */
   CEESGL(&condtok,&qdata,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEESGL failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
  /* .
     .
     . */
}
```

2. Following is an example of CEESGL called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTSGL
      ************************************************
      **                                            **
      ** CBLSGL - Call CEESGL to signal a condition **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSGL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  CONDTOK.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       01  QDATA                   PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       01  SEV                     PIC S9(4) BINARY.
       01  MSGNO                   PIC S9(4) BINARY.
       01  CASE                    PIC S9(4) BINARY.
       01  SEV2                    PIC S9(4) BINARY.
       01  CNTRL                   PIC S9(4) BINARY.
       01  FACID                   PIC X(3).
       01  ISINFO                  PIC S9(9) BINARY.

       PROCEDURE DIVISION.
       PARA-CBLSGL.
      ************************************************
      ** Call CEENCOD with the values assigned above
      ** to build a condition token "CONDTOK"
```

```
     ** Set CONDTOK to sev = 0, msgno = 1 facid = CEE
     ************************************************
          MOVE 0 TO SEV.
          MOVE 1 TO MSGNO.
          MOVE 1 TO CASE.
          MOVE 0 TO SEV2.
          MOVE 1 TO CNTRL.
          MOVE "CEE" TO FACID.
          MOVE 0 TO ISINFO.

          CALL "CEENCOD" USING SEV, MSGNO, CASE,
              SEV2, CNTRL, FACID, ISINFO, CONDTOK, FC.
          IF NOT CEE000 of FC  THEN
              DISPLAY "CEENCOD failed with msg "
                  Msg-No of FC UPON CONSOLE
              STOP RUN
          END-IF.

     ** Call CEESGL to signal the condition with
     ************************************************
     **     the condition token and qdata described
     **     in CONDTOK and QDATA
     ************************************************
          MOVE 0 TO QDATA.
          CALL "CEESGL" USING CONDTOK, QDATA, FC.
          IF NOT CEE000 of FC  THEN
              DISPLAY "CEESGL  failed with msg "
                  Msg-No of FC UPON CONSOLE
              STOP RUN
          END-IF.

          GOBACK.
```

3. Following is an example of CEESGL called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMSGL                         */
 /****************************************************/
 /**                                                 */
 /** Function: CEESGL  - Signal a Condition          */
 /**                                                 */
 /****************************************************/
 PLISGL: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL 01 CONDTOK,                /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
    DCL QDATA  REAL FIXED BINARY(31,0);
    DCL 01 FC,                     /* Feedback token */
           03 MsgSev    REAL FIXED BINARY(15,0),
           03 MsgNo     REAL FIXED BINARY(15,0),
           03 Flags,
              05 Case      BIT(2),
              05 Severity  BIT(3),
              05 Control   BIT(3),
           03 FacID     CHAR(3),      /* Facility ID */
           03 ISI   /* Instance-Specific Information */
                        REAL FIXED BINARY(31,0);
```

```
   /* Give CONDTOK value of condition CEE001        */
   ADDR( CONDTOK ) -> CEEIBMCT = CEE001;

   /* Signal condition CEE001 with qualifying data  */
   QDATA = 1;
   CALL CEESGL ( CONDTOK, QDATA, FC );
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST( 'Condition CEE001 signalled' );
      END;
   ELSE  DO;
      DISPLAY( 'CEESGL  failed with msg '
         || FC.MsgNo );
      STOP;
      END;

END PLISGL;
```

CEESTXF—Transform string characters into collation weights

CEESTXF, which is analogous to the C language function `strcfrm()`, transforms every character in a character string to its unique collation weight. The collation weights are established from the LC_COLLATE category for the locale. CEESTFX also returns the length of the transformed string. CEESTXF is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
►►──CEESTXF──(──omitted_parm──,──mbstring──,──number──,──txfstring──,──────────►

►─length──,──fc──)──────────────────────────────────────────────────────────────►◄
```

omitted_parm
: This parameter is reserved for future expansion and must be omitted. For information about how to code an omitted parm, see “Invoking callable services” on page 114.

mbstring **(input)**
: A halfword length-prefixed character string (VSTRING) that is to be transformed.

number **(input)**
: A 4-byte integer that specifies the number of bytes of *mbstring* to be transformed. The value of this parameter must be greater than zero; otherwise, an error is reported, and no transformation is attempted.

txfstring **(output)**
: A halfword length-prefixed character string (VSTRING) where the transformation of *mbstring* is to be placed.

length **(output)**
: A 4-byte integer that specifies the length of the transformed string, if successful.

fc **(output)**
: A 12-byte feedback code, optional in some languages, that indicates the result

of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.
CEE3TF	3	4015	Input Error: The number of characters to be transformed must be greater than zero.

Usage notes

- PL/I MTF consideration—CEESETL is not supported in PL/I MTF applications.
- This callable service uses the C/C++ runtime library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- For more information about the `setlocale()` ,see "COUNTRY" on page 22, "CEE3CTY—Set default country" on page 133, and "CEE3LNG—Set national language" on page 172.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 411.

Examples

1. Following is an example of CEESTXF called by COBOL.

```
 CBL LIB,QUOTE
*Module/File Name: IGZTSTXF
*************************************************
* Example for callable service CEESTXF         *
*  COBSTXF - Query current collate category and *
*            build input string as function of *
*            locale name.                      *
*           Translate string as function of    *
*            locale.                           *
*  Valid only for COBOL for MVS & VM Release 2  *
*  or later.                                   *
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBSTXF.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  MBS.
     02  MBS-Length  PIC S9(4) BINARY.
     02  MBS-String  PIC X(10).
 01  TXF.
     02  TXF-Length  PIC S9(4) BINARY.
     02  TXF-String  PIC X(256).
 01  Locale-Name.
     02  LN-Length  PIC S9(4) BINARY.
     02  LN-String  PIC X(256).
* Use Locale category constants
 COPY CEEIGZLC.
 01  MBS-Size    PIC S9(9)  BINARY  VALUE 0.
 01  TXF-Size    PIC S9(9)  BINARY  VALUE 0.
 01  FC.
     02  Condition-Token-Value.
     COPY CEEIGZCT.
         03  Case-1-Condition-ID.
```

```
                   04  Severity    PIC S9(4) BINARY.
                   04  Msg-No      PIC S9(4) BINARY.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) BINARY.
                   04  Cause-Code  PIC S9(4) BINARY.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) BINARY.
       PROCEDURE DIVISION.
      **************************************************
      *  Call CEEQRYL to retrieve locale name
      **************************************************
           CALL "CEEQRYL" USING LC-COLLATE,
                                Locale-Name, FC.
      **************************************************
      *  Check feedback code and set input string
      **************************************************
           IF Severity = 0
              IF LN-String (1:LN-Length) =
                    "Sv-SE.IBM-1047"
                 MOVE 10 TO MBS-Length
                 MOVE 10 TO MBS-Size
                 MOVE "7,123,456."
                         TO MBS-String (1:MBS-Length)
              ELSE
                 MOVE 7 TO MBS-Length
                 MOVE 7 TO MBS-Size
                 MOVE "8765432"
                        TO MBS-String (1:MBS-Length)
              END-IF
           ELSE
              DISPLAY "Call to CEEQRYL failed. " Msg-No
              STOP RUN
           END-IF.
                MOVE SPACES TO TXF-String.
           MOVE 0 to TXF-Length.

      **************************************************
      *  Call CEESTXF to translate the string
      **************************************************
           CALL "CEESTXF" USING OMITTED, MBS, MBS-Size,
                                TXF, TXF-Size, FC.
     **************************************************
      *  Check feedback code and return length
      **************************************************
           IF Severity = 0
              IF TXF-Length > 0
                 DISPLAY "Translated string is "
                         TXF-String
              ELSE
                 DISPLAY "String not translated."
              END-IF
           ELSE
              DISPLAY "Call to CEESTXF failed. " Msg-No
           END-IF.
          STOP RUN.
       END PROGRAM COBSTXF.
```

2. Following is an example of CEESTXF called by PL/I.

```
*PROCESS MACRO;
 /*Module/File Name: IBMSTXF                        */
 /***************************************************/
 /* Example for callable service CEESTXF            */
 /* Function: Query current collate category and    */
 /*   build input string as function of locale name. */
 /*   Translate string as function of locale.       */
```

```
/****************************************************/
PLISTXF: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs       */
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants   */
%INCLUDE CEEIBMLC; /* Locale category constants     */
/* CEESTXF service call arguments */
DCL MBSTRING  CHAR(10) VARYING;   /* input string   */
DCL MBNUMBER  BIN FIXED(31);      /* input length   */
DCL TXFSTRING CHAR(256) VARYING;  /* output string  */
DCL TXFLENGTH BIN FIXED(31);      /* output length  */
/* CEEQRYL service call arguments */
DCL LOCALE_NAME_COLLATE CHAR(256) VARYING;
DCL 01 FC,                        /* Feedback token */
       03 MsgSev    REAL FIXED BINARY(15,0),
       03 MsgNo     REAL FIXED BINARY(15,0),
       03 Flags,
          05 Case      BIT(2),
          05 Severity  BIT(3),
          05 Control   BIT(3),
       03 FacID     CHAR(3),       /* Facility ID */
       03 ISI      /* Instance-Specific Information */
                    REAL FIXED BINARY(31,0);
  /* retrieve active locale for collate category */
  /* Use LC_COLLATE category const from CEEIBMLC */
  CALL CEEQRYL ( LC_COLLATE, LOCALE_NAME_COLLATE, FC);
  /* FBCHECK macro used (defined in CEEIBMCT) */
  IF FBCHECK( FC, CEE000 ) THEN
    DO; /* successful query, set string for CEESTXF */
      IF LOCALE_NAME_COLLATE = 'Sv_SE.IBM-1047' THEN
          MBSTRING = '7,123,456.';
      ELSE
          MBSTRING = '8765432';
      MBNUMBER = LENGTH(MBSTRING);
    END;
  ELSE
    DO;
      DISPLAY ( 'Locale LC_COLLATE  '||FC.MsgNo );
      STOP;
    END;
  TXFSTRING = ';
  CALL CEESTXF ( *, MBSTRING, MBNUMBER,
                    TXFSTRING, TXFLENGTH, FC );
  IF FBCHECK( FC, CEE000 ) THEN
    DO; /* successful call, use transformed length */
      IF TXFLENGTH >0 THEN
        DO;
          PUT SKIP LIST( 'Transformed string is '||
                  SUBSTR(TXFSTRING,1, TXFLENGTH) );
        END;
    END;
  ELSE
    DO;
      IF FBCHECK( FC, CEE3TF ) THEN
        DO;
          DISPLAY ( 'Zero length input string' );
        END;
    END;
END PLISTXF;
```

CEETDLI—Invoke IMS

CEETDLI provides an interface to PL/I facilities that operates in IMS. In assembler, COBOL, PL/I, and C/C++, you can also invoke IMS by using the following interfaces:

- In assembler, the ASMTDLI interface

- In COBOL, the CBLTDLI interface
- In PL/I, the PLITDLI interface
- In C/C++, the CTDLI interface, a `ctdli()` function call

CEETDLI performs essentially the same functions as these interfaces, but it offers some advantages, particularly if you plan to run an ILC application in IMS.

The names CEETDLI, AIBTDLI, ASMTDLI, CBLTDLI, CTDLI, and PLITDLI are all interpreted as IMS interfaces. If you are currently using them in any other way in your application, you must change them. The CEETDLI interface supports calls to IMS that use an application interface block (AIB) or a program communication block (PCB).

Syntax

```
►►──CEETDLI──(──┬──────────────┬──,──function──,──┬──────┬──)──────────────►◄
                └─parmcount──,─┘                  └─args─┘
```

parmcount
: A fullword integer specifying the total number of arguments for the CEETDLI call. This option usually is not needed and can be omitted; it is supported for compatibility with earlier interface modules.

function
: The IMS function that you want to perform. The possible values for this field are defined by IMS, not Language Environment.

args
: Arguments that you pass to IMS. You cannot pass runtime options as CEETDLI arguments. You cannot alter the settings of runtime options when invoking IMS facilities. The order and meaning of the arguments is defined by IMS, not Language Environment.

Usage notes

- CEETDLI is not supported with CICS.
- z/OS UNIX considerations—IMS supports only applications that use the POSIX(ON) runtime option from a single thread. Calls to z/OS UNIX threading functions are restricted under IMS. See *z/OS XL C/C++ Programming Guide* for a list of restrictions on running IMS and z/OS UNIX.

For more information

- For more information about AIB and a complete description of all available IMS functions and argument parameters you can specify in CEETDLI, see *IMS Application Programming Guide*.
- For more information about CEETDLI in the context of other DLI interfaces, and in the context of IMS condition handling, see *z/OS Language Environment Programming Guide*.

Examples

1. Following is an example of CEETDLI called by C.

```
/*Module/File Name: EDCMRCR   */

#pragma runopts(env(IMS),plist(IMS))
```

```
#include <ims.h>
#include <leawi.h>
#define io_pcb ((IO_PCB_TYPE *)(__pcblist??(0??)))

/* ------------------------------------------------ */
/*                                                  */
/*   Function: Use CEETDLI - interface to IMS       */
/*                        from C.                   */
/*                                                  */
/*   In this example, a call is made to CEETDLI     */
/*   to interface to IMS for an IMS service.        */
/*                                                  */
/* ------------------------------------------------ */
/*                ENTRY POINT                       */
/* ------------------------------------------------ */

main() {

  static char func_GU??(4??) = "GU  ";
  char msg_seg_io_area??(32??);
  CEETDLI(func_GU,io_pcb,msg_seg_io_area);
}
```

2. Following is an example of CEETDLI called by COBOL.

```
      *Module/File Name: IGZTTDLI
      ***********************************************/
      **                                            */
      ** Function: Use CEETDLI - interface to IMS   */
      **                        from COBOL.         */
      **                                            */
      ** In this example, a call is made to CEETDLI */
      ** to interface to IMS for an IMS service.    */
      **                                            */
      ***********************************************/
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL2IMS.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  FUNC-GU             PIC X(4) VALUE "GU  ".
       01  MSG-SEG-IO-AREA     PIC X(32).
       LINKAGE SECTION.
       01  IO-PCB              PIC X(48).
      ** ENTRY POINT:                               */
       PROCEDURE DIVISION USING IO-PCB.
           CALL "CEETDLI" USING FUNC-GU,IO-PCB,
                                MSG-SEG-IO-AREA.
           GOBACK.
```

3. Following is an example of CEETDLI called by PL/I.

```
*PROCESS MACRO SYSTEM(IMS);
 /*Module/File Name: IBMTDLI                         */
 /* ------------------------------------------ */
 /*                                            */
 /* Function: Use CEETDLI - interface to IMS   */
 /*                        from PL/I.          */
 /*                                            */
 /* In this example, a call is made to CEETDLI */
 /* to interface to IMS for an IMS service.    */
 /*                                            */
 /* ------------------------------------------ */
 /*               ENTRY POINT                  */
 /* ------------------------------------------ */
 PLI2IMS: PROCEDURE(IO_PTR) OPTIONS(MAIN NOEXECOPS);

 %INCLUDE   CEEIBMAW;

 DCL   FUNC_GU         CHAR(4)   INIT('GU  ');
DCL   IO_PTR          PTR;
DCL 1 IO_PCB          CHAR(48) BASED (IO_PTR);

DCL 1 MSG_SEG_IO_AREA CHAR(32);

CALL CEETDLI (FUNC_GU, IO_PCB, MSG_SEG_IO_AREA);

RETURN;

END PLI2IMS;
```

CEETEST—Invoke Debug Tool

CEETEST invokes a debug service such as Debug Tool, which is supplied with z/OS.

Debug Tool supports debugging of Language Environment, except for some noted restrictions. For more information about Debug Tool for z/OS, see http://www.ibm.com/software/awdtools/debugtool/. If you want to invoke another interactive debug service, refer to the appropriate user's guide.

Syntax

```
►►──CEETEST──(──┬─────────────────────────┬─,──fc──)────────────────────────►◄
                └─string_of_commands──,───┘
```

string_of_commands **(input)**
A halfword-prefixed string containing a debug tool command list. *string_of_commands* is optional. If a debug tool is available, the commands in the list are passed to the debug tool and carried out. If this parameter is omitted, your debug service defines the action taken. For more information, refer to the appropriate user's guide for your debug service.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE2F2	3	2530	A debug tool was not available.
CEE2F7	3	2535	Profiler loaded; a debug tool was not available.

Usage notes

- z/OS UNIX considerations—CEETEST applies to the enclave. All threads in the enclave can access debugger information.

For more information

- If you are using CEETEST to invoke Debug Tool and need more information about how to create a Debug Tool command list, refer to http://www.ibm.com/software/awdtools/debugtool/..

Examples

1. Following is an example of CEETEST called by C/C++.

```
/*Module/File Name: EDCTEST   */

#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main (void) {

   int x,y,z;
   _VSTRING commands;
   _FEEDBACK fc;

   strcpy(commands.string,
           "AT LINE 30 { LIST(x); LIST(y); GO; }");
   commands.length = strlen(commands.string);

   CEETEST(&commands,&fc);

   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEETEST failed with message number %d\n",
              fc.tok_msgno);
      exit(2999);
   }
   x = y = 12;
  /* .
      .
      . */
   /* debug tool displays the values of x and y */
   /* at statement 30 */
  /* .
      .
      . */
}
```

2. Following is an example of CEETEST called by COBOL.

```
CBL LIB,QUOTE
      *Module/File Name: IGZTTEST
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IBCT002.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  MANVAR1                PIC S9(9) BINARY.
       01  CEETEST-PARMS.
           02  Vstring-length     PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char   PIC X,
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                              of CEETEST-PARMS.
           01 FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity   PIC S9(4) BINARY.
                   04  Msg-No     PIC S9(4) BINARY.
```

```
            03  Case-2-Condition-ID
                      REDEFINES Case-1-Condition-ID.
                04  Class-Code  PIC S9(4) BINARY.
                04  Cause-Code  PIC S9(4) BINARY.
            03  Case-Sev-Ctl    PIC X.
            03  Facility-ID     PIC XXX.
        02  I-S-Info            PIC S9(9) BINARY.
    PROCEDURE DIVISION.
    PARA-IBCT002.
        MOVE 0 TO MANVAR1
        COMPUTE MANVAR1 = MANVAR1 + 100
        DISPLAY "The value of MANVAR1 is " , MANVAR1
   ***************************************************
   *  CALL CEETEST FOR FIRST  TIME.                  *
   ***************************************************
        MOVE 70 TO Vstring-length of CEETEST-PARMS.
        MOVE "DESC PROGRAM AT ENTRY IBCT002:>SUBRTN"
          TO Vstring-text of CEETEST-PARMS(1:37).
        move " PERFORM Q LOC GO END-PERFORM GO "
          TO Vstring-text of CEETEST-PARMS(38:33).
        CALL "CEETEST" USING CEETEST-PARMS, FC.
        IF NOT CEE000 of FC  THEN
            DISPLAY "CEETEST(1st call) failed with msg "
                Msg-No of FC UPON CONSOLE
            STOP RUN
        END-IF.
   ***************************************************
   *  CALL CEETEST A SECOND TIME.                    *
   ***************************************************
        MOVE 4 TO Vstring-length of CEETEST-PARMS.
        MOVE "QUIT "
          TO Vstring-text of CEETEST-PARMS.
        CALL "CEETEST" USING CEETEST-PARMS, FC.
        IF NOT CEE000 of FC  THEN
            DISPLAY "CEETEST(2nd call) failed with msg "
                Msg-No of FC UPON CONSOLE
            STOP RUN
        END-IF.
   **************************************************
        GOBACK.
   **************************************************
    IDENTIFICATION DIVISION.
    PROGRAM-ID. SUBRTN.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  MANVAR1                 PIC S9(9) BINARY.
    01  MVAR                    PIC S9(9) BINARY.
   PROCEDURE DIVISION.
    PARA-SUBRTN.
        COMPUTE MVAR = MVAR + 100 .
        COMPUTE MANVAR1 = MANVAR1 + 100 .

        GOBACK.
    END PROGRAM SUBRTN.
    END PROGRAM IBCT002.
```

3. Following is an example of CEETEST called by PL/I.

```
*PROCESS MACRO;
 /* Module/File Name: IBMTEST                       */
 /**************************************************/
 /**                                              **/
 /** Function: CEETEST - Invoke a Debug Tool      **/
 /**                                              **/
 /**************************************************/
 PLITEST: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
```

```
   %INCLUDE  CEEIBMCT;

   DCL DBGCMD  CHAR(255) VARYING;
   DCL 01 FC,                    /* Feedback token */
          03 MsgSev    REAL FIXED BINARY(15,0),
          03 MsgNo     REAL FIXED BINARY(15,0),
          03 Flags,
             05 Case      BIT(2),
             05 Severity  BIT(3),
             05 Control   BIT(3),
          03 FacID     CHAR(3),      /* Facility ID */
          03 ISI   /* Instance-Specific Information */
                       REAL FIXED BINARY(31,0);

   DBGCMD = 'QUERY PROGRAMMING LANGUAGE';

   CALL CEETEST ( DBGCMD, FC );
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST('Debug tool called with command: '
         || DBGCMD );
      END;
   ELSE  DO;
      DISPLAY('CEETEST failed with msg ' || FC.MsgNo);
      STOP;
      END;

END PLITEST;
```

CEEUTC—Get coordinated universal time

CEEUTC is an alias of CEEGMT. See “CEEGMT—Get current Greenwich Mean Time” on page 292 for more information.

Chapter 6. Bit manipulation routines

This topic lists the Language Environment bit manipulation routines. Bits are numbered from right to left, starting from 0.

CEESICLR—Bit clear

Purpose

CEESICLR returns a copy of its *parm1* input, but with one bit selectively set to 0.

Syntax

```
►►—CEESICLR—(—parm1—,—parm2—,—fc—,—result—)——————————————————————►◄
```

parm1 **(input)**
The first input to the Bit Clear routine. The input can be any 32-bit integer.

parm2 **(input)**
The second input to the Bit Clear routine. The *parm2* value is a 32-bit integer in the range between 0 and 31, inclusive.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0		The service completed successfully.
CEE1VC	2	2028	The value of the second argument was outside the valid range *range* in math routine *routine-name*.

result **(output)**
The result of the Bit Clear routine. The output is a copy of *parm1*, but with the bit numbered *parm2* (counting from the right) set to 0.

CEESISET—Bit set

Purpose

CEESISET returns a copy of its *parm1* input, but with one bit selectively set to 1.

Syntax

```
►►──CEESISET──(──parm1──,──parm2──,──fc──,──result──)──────────────────►◄
```

parm1 **(input)**
The first input to the Bit Set routine. The input can be any 32-bit integer.

parm2 **(input)**
The second input to the Bit Set routine. The *parm2* value is a 32-bit integer in the range between 0 and 31, inclusive.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0		The service completed successfully.
CEE1VC	2	2028	The value of the second argument was outside the valid range *range* in math routine *routine-name*.

result **(output)**
The result of the Bit Set routine. The output is a copy of *parm1*, but with the bit numbered *parm2* (counting from the right) set to 1.

CEESISHF—Bit shift

Purpose

CEESISHF returns a copy of its *parm1* input right- or left-shifted by the number of bits indicated by *parm2*.

Syntax

```
►►──CEESISHF──(──parm1──,──parm2──,──fc──,──result──)──────────────────►◄
```

parm1 **(input)**
The first input to the Bit Shift routine. The input can be any 32-bit integer.

parm2 **(input)**
The second input to the Bit Shift routine. The *parm2* value is a 32-bit integer in the range between -32 and 32, inclusive.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0		The service completed successfully.
CEE1VC	2	2028	The value of the second argument was outside the valid range *range* in math routine *routine-name*.

result **(output)**
The result of the Bit Shift routine. The output is a 32-bit integer whose value depends upon the value of *parm2*; in either case, vacated bits are set to 0:

- If *parm2* is greater than or equal to 0, *result* is a copy of *parm1* shifted left by *parm2* bits.
- If *parm2* is less than 0, *result* is a copy of *parm1* shifted right by |*parm2*| bits.

CEESITST—Bit test

Purpose

CEESITST selectively tests a bit in its *parm1* input to determine if the bit is on.

Syntax

```
►►—CEESITST—(—parm1—,—parm2—,—fc—,—result—)——————————————————————►◄
```

parm1 **(input)**
The first input to the Bit Test routine. The input can be any 32-bit integer.

parm2 **(input)**
The second input to the Bit Test routine. The *parm2* value is a 32-bit integer in the range between 0 and 31, inclusive.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0		The service completed successfully.
CEE1VC	2	2028	The value of the second argument was outside the valid range *range* in math routine *routine-name*.

result **(output)**
The result of the Bit Test routine. The output is a 32-bit integer with one of the following values; bits are counted from the right:

1	if bit number *parm2* in *parm1* is 1
0	if bit number *parm2* in *parm1* is 0

Chapter 7. Language Environment math services

Language Environment math services provide standard math computations. You can call them from Language Environment-conforming languages or from assembler routines by using the call interface, or the syntax specific to the HLL of your application.

Math services do not depend on enclave-level resources. You can invoke them from any thread.

If your application uses extended-precision arithmetic and runs on a 370-mode machine under VM, you must specify the TRAP(ON) runtime option (the default) and add the CMSLIB TXTLIB with the GLOBAL TXTLIB command.

Call interface to math services

The syntax for math services has two forms, depending on how many input parameters the routine requires. The first four letters of the math services are always CEES. The fifth character is *x*, which you replace according to the parameter types listed in "Parameter types: parm1 and parm2." The last three letters name the math function performed. In the following examples, the first function performed is the absolute value (ABS), and the second function is the positive difference (DIM).

One Parameter

```
►►──CEESxABS──(──parm1──,──fc──,──result──)──────────────────────►◄
```

Two Parameters

```
►►──CEESxDIM──(──parm1──,──parm2──,──fc──,──result──)────────────►◄
```

Parameter types: parm1 and parm2

The first parameter (*parm1*) is mandatory. The second parameter (*parm2*) is used only when you use a math service with two parameters. The *x* in the fifth space of CEES*x* must be replaced by a parameter type for input and output. Substitute I, S, D, Q, T, E, or R for *x*:

I	32-bit binary integer
S	32-bit single floating-point number
D	64-bit double floating-point number
Q	128-bit extended floating-point number

T	32-bit single floating-point complex number. This parameter type consists of a real part and an imaginary part, each of which is a 32-bit single floating-point number.
E	64-bit double floating-point complex number. This parameter type consists of a real part and an imaginary part, each of which is a 64-bit double floating-point number.
R	128-bit extended floating-point complex number. This parameter type consists of a real part and an imaginary part, each of which is a 128-bit extended floating-point number.

Language Environment math services expect normalized input.

In the described routines, the output range for complex-valued functions can be determined from the input range. For functions of complex variables, the image of the input is generally a non-rectangular shape. For this reason, the output range is not provided .

Feedback code parameter (fc)

The *fc* value is a feedback code that indicates the result of the math service. If you specify *fc* as an argument, feedback information in the form of a condition token is returned to the calling routine. The condition token indicates whether the routine completed successfully or whether a condition was encountered while the routine was running. If you do not specify *fc* as an argument and the requested service does not successfully complete, the condition is signaled. Math services call other services that might generate feedback codes.

Language-specific built-in math services

C/C++, COBOL, Fortran, and PL/I offer built-in math services that you can also use under Language Environment. For a description of these functions, refer to the reference documentation associated with each language.

Calls to math services from different languages

Table 28 shows sample calls to the Language Environment math service CEESLOG—logarithm base e, as made from C/C++, COBOL, and PL/I. For more examples, see "Examples of math services" on page 475.

Table 28. Examples of calls to CEESLOG

Called from	Code example

C/C++	`#include <leawi.h>`
	`#include <string.h>`
	`#include <stdio.h>`
	
	`int main (void) {`
	
	`  float int1, intr;`
	
	`  _FEEDBACK fc;`
	`  #define SUCCESS "\0\0\0\0"`
	
	`  int1 = 39;`
	`  CEESSLOG(&int1,&fc,&intr);`
	
	`  if (memcmp(&fc,SUCCESS,4) != 0) {`
	`     printf("CEESSLOG failed with message number %d\n",`
	`            fc.tok_msgno);`
	`     exit(2999);`
	`  }`
	
	`  printf("Log base e of %f is %f\n",int1,intr);`
	`}`

COBOL	⋮
	`    77   ARG1RS  COMP-1.`
	`    77   FBCODE  PIC X(12).`
	`    77   RESLTRS COMP-1.`
	
	`         CALL "CEESSLOG" USING ARG1RS , FBCODE , RESLTRS.`
	⋮

PL/I	⋮
	`DCL ARG1 RESULT REAL FLOAT DEC (6);`
	`DCL FC   CHARACTER (12);`
	
	`CALL CEESSLOG (ARG1, FC, RESULT)`
	⋮

Math services

This section describes the Language Environment math services. It also provides examples of how various services are called by different programming languages.

CEESxABS—Absolute value

CEESxABS returns the absolute value of the parameter by using the equation: *result* = |*parm1*|

The following routines are provided for the various data types supported:

CEESIABS
: 32-bit binary integer

CEESSABS
: 32-bit single floating-point number

CEESDABS
: 64-bit double floating-point number

CEESQABS
: 128-bit extended floating-point number

CEESTABS
: 32-bit single floating-point complex number

CEESEABS
: 64-bit double floating-point complex number

CEESRABS
: 128-bit extended floating-point complex number

Syntax

```
►►—CEESxABS—(—parm1—,—fc—,—result—)——————————————————————►◄
```

parm1 **(input)**
The input to the absolute value routine. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1V9	1	2025	An underflow occurred in math routine *routine-name*.

result **(output)**
The result of the absolute value routine. The output range is the non-negative numbers:

$\leq \Omega$

Omega varies depending on the precision of *parm1*:

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxACS—Arccosine

CEESxACS returns the arccosine of the parameter by using the equation: *result* = arccos (*parm1*)

The following routines are provided for the various data types supported:

CEESSACS
32-bit single floating-point number

CEESDACS
64-bit double floating-point number

CEESQACS
128-bit extended floating-point number

Syntax

```
►►──CEESxACS──(──parm1──,──fc──,──result──)──────────────────────────────►◄
```

parm1 **(input)**
The input to the arccosine routine. The input range is:

$|parm1| \leq 1$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the parameter was greater than *limit* in math routine *routine-name*.

result **(output)**
The result, in radians, of the arccosine routine.

$0 \le result \le \pi$

CEESxASN—Arcsine

CEESxASN returns the arcsine of the parameter by using the equation: *result* = arcsin(*parm1*)

The following routines are provided for the various data types supported:

CEESSASN
32-bit single floating-point number

CEESDASN
64-bit double floating-point number

CEESQASN
128-bit extended floating-point number

Syntax

```
►►──CEESxASN──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**
The input to the arcsine routine. The input range is:

$|parm1| \le 1$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the parameter was greater than *limit* in math routine *routine-name*.

result **(output)**
The result, in radians, of the arcsine routine. The output range is:

$|result| < \Pi/2$

CEESxATH—Hyperbolic arctangent

CEESxATH returns the hyperbolic arctangent of the parameter by using the following equation:

$$\mathit{result} = \tanh^{-1}(\mathit{parm}1)$$

The following routines are provided for the various data types supported:

CEESSATH
32-bit single floating-point number

CEESDATH
64-bit double floating-point number

CEESQATH
128-bit extended floating-point number

CEESTATH
32-bit single floating-point complex number

CEESEATH
64-bit double floating-point complex number

CEESRATH
128-bit extended floating-point complex number

Syntax

```
►►──CEESxATH──(──parm1──,──fc──,──result──)─────────────────────────────►◄
```

parm1 **(input)**
The input to the hyperbolic arctangent routine. The input range for real variables is:

$$|\mathit{parm1}| \leq 1$$

For complex variables, *parm1* cannot be equal to 1 or -1.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

Code	Severity	Message number	Message text
CEE1V1	2	2017	The absolute value of the argument was greater than or equal to *limit* in math routine *routine-name*.
CEE1V6	2	2022	The value of the argument was plus or minus *limit* in math routine *routine-name*.

result **(output)**
The result of the hyperbolic arctangent routine. The output range for functions of real variables is:

$|result| \leq \Omega$

Omega varies, depending on the precision of *parm1*:

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxATN—Arctangent

CEESxATN returns the arctangent of the parameter by using the equation : *result* = arctan(*parm1*)

The following routines are provided for the various data types supported:

CEESSATN
32-bit single floating-point number

CEESDATN
64-bit double floating-point number

CEESQATN
128-bit extended floating-point number

CEESTATN
32-bit single floating-point complex number

CEESEATN
64-bit double floating-point complex number

CEESRATN
128-bit extended floating-point complex number

Syntax

```
►►──CEESxATN──(──parm1──,──fc──,──result──)─────────────────────────────►◄
```

parm1 **(input)**
The input to the arctangent routine. The input range for real variables is not

restricted. The input range of complex variables in *parm1* is not equal to i or -i, where:

$i = \sqrt{-1}$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1V6	2	2022	The value of the parameter was plus or minus *limit* in math routine *routine-name*.
CEE1V9	1	2025	An underflow occurred in math routine *routine-name*.

result **(output)**
The result, in radians, of the arctangent routine. The output range for functions of real variables is:

$|result| < \Pi/2$

CEESxAT2—Arctangent2

CEESxAT2 calculates a result by using the equation: *result* = the angle (in radians) between the positive X axis and a vector defined by (*parm2*, *parm1*) with a range from:

$-\Pi$ to Π, with Π,

For example, if *parm1* and *parm2* are positive, then *result* = arctan (*parm1*/*parm2*).

The following routines are provided for the various data types supported:

CEESSAT2
32-bit single floating-point number

CEESDAT2
64-bit double floating-point number

CEESQAT2
128-bit extended floating-point number

Syntax

```
▸▸──CEESxAT2──(──parm1──,──parm2──,──fc──,──result──)──────────────────────▸◂
```

parm1 **(input)**
The first input to the arctangent2 routine. The input range of *parm1* cannot equal 0 if *parm2* equals 0.

parm2 **(input)**
The second parameter to the arctangent2 routine. The input range of *parm2* cannot equal 0 if *parm1* equals 0.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UU	2	2014	Both parameters were equal to *limit* in math routine *routine-name*.
CEE1V9	1	2025	An underflow occurred in math routine *routine-name*.

result **(output)**
The result, in radians, of the arctangent2 routine. The output range is:

$|result| \leq \Pi$

CEESxCJG—Conjugate of complex

CEESxCJG returns the conjugate of the complex number by using the equation: *result* = *u* - *vi*, where *parm1* = *u* + *vi*.

The following routines are provided for the various data types supported:

CEESTCJG
32-bit single floating-point complex number

CEESECJG
64-bit double floating-point complex number

CEESRCJG
128-bit extended floating-point complex number

Syntax

```
▸▸—CEESxCJG—(—parm1—,—fc—,—result—)——————————————————————▸◂
```

parm1 **(input)**
The input to the math service. Any representable complex number can be used as input.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the conjugate of complex routine.

CEESxCOS—Cosine

CEESxCOS returns the cosine of the parameter by using the equation: *result* = cos(*parm1*)

The following routines are provided for the various data types supported:

CEESSCOS
32-bit single floating-point number

CEESDCOS
64-bit double floating-point number

CEESQCOS
128-bit extended floating-point number

CEESTCOS
32-bit single floating-point complex number

CEESECOS
64-bit double floating-point complex number

CEESRCOS
128-bit extended floating-point complex number

Syntax

```
▸▸—CEESxCOS—(—parm1—,—fc—,—result—)——————————————————————▸◂
```

parm1 **(input)**
The type is determined by the fifth character of the service name. The input range for real variables varies:

For	Input range
Single floating-point numbers:	$\|parm1\| < (2^{18} \bullet \Pi)$
Double floating-point numbers:	$\|parm1\| < (2^{50} \bullet \Pi)$
Extended floating-point numbers:	$\|parm1\| < 2^{100}$

For complex functions, the input range differs for the imaginary and real parts of the input.

For ...	Input range
The imaginary part:	$\| \mathrm{Im}\ (parm1)\| < 174.673$
The real part: single floating-point complex numbers:	$\| \mathrm{Re}\ (parm1)\| < (2^{18} \bullet \Pi)$
The real part: double floating-point complex numbers:	$\| \mathrm{Re}\ (parm1)\| < (2^{50} \bullet \Pi)$
The real part: extended floating-point complex numbers:	$\| \mathrm{Re}\ (parm1)\| < 2^{100}$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UT	2	2013	The absolute value of the imaginary part of the parameter was greater than *limit* in math routine *routine-name*.
CEE1V1	2	2017	The absolute value of the parameter was greater than or equal to *limit* in math routine *routine-name*.
CEE1V3	2	2019	The absolute value of the real part of the parameter was greater than or equal to *limit* in math routine *routine-name*.

result **(output)**
The result of the cosine routine. The output range for functions of real variables is:

$|result| \leq 1$

CEESxCSH—Hyperbolic cosine

CEESxCSH returns the hyperbolic cosine of the parameter by using the equation: *result* = cosh(*parm1*)

The following routines are provided for the various data types supported:

CEESSCSH
: 32-bit single floating-point number

CEESDCSH
: 64-bit double floating-point number

CEESQCSH
: 128-bit extended floating-point number

CEESTCSH
: 32-bit single floating-point complex number

CEESECSH
: 64-bit double floating-point complex number

CEESRCSH
: 128-bit extended floating-point complex number

Syntax

```
►►──CEESxCSH──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**

The input to the hyperbolic cosine routine. The input range varies, depending on the function and type of number:

For ...	Input range
Functions of real variables:	$\|parm1\| < 175.366$
The real part of complex numbers:	$\|Re\ (parm1)\| < 174.673$
The imaginary part of complex numbers: single floating-point complex numbers:	$\|Im\ (parm1)\| < (2^{18} \bullet \Pi)$
The imaginary part of complex numbers: double floating-point complex numbers:	$\|Im\ (parm1)\| < (2^{50} \bullet \Pi)$
The imaginary part of complex numbers: extended floating-point complex numbers:	$\|Im\ (parm1)\| < 2^{100}$

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the parameter was greater than *limit* in math routine *routine-name*.

result **(output)**

The result of the hyperbolic cosine routine. The output range for functions of real variables varies:

For ...	Output range
Single-precision routines:	$1 \leq \mathit{result} \leq \Omega$
Double-precision routines:	$\Omega = 16^{63}(1 - 16^{6})$ $16^{63}(1 - 16^{-14})$
Extended-precision routines:	$16^{63}(1 - 16^{-28})$

CEESxCTN—Cotangent

CEESxCTN returns the cotangent of the parameter by using the equation: *result* = cot(*parm1*)

The following routines are provided for the various data types supported:

CEESSCTN
: 32-bit single floating-point number

CEESDCTN
: 64-bit double floating-point number

CEESQCTN
: 128-bit extended floating-point number

Syntax

```
►►──CEESxCTN──(──parm1──,──fc──,──result──)─────────────────────────────►◄
```

parm1 **(input)**
The input, in radians, into the cotangent routine. The input range varies, depending on the value of *parm1*:

If *parm1* is a 32-bit single floating-point number:	$\lvert parm1 \rvert < (2^{18} \bullet \Pi)$
If *parm1* is a 64-bit double floating-point number:	$\lvert parm1 \rvert < (2^{50} \bullet \Pi)$
If *parm1* is a 128-bit extended floating-point number:	$\lvert parm1 \rvert < 2^{100}$

If this is an extended floating-point number, this argument cannot approach a multiple of pi. Single floating-point numbers and double floating-point numbers cannot approach zero.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UI	2	2002	The parameter value was too close to one of the singularities (plus or minus pi/2, plus or minus 3pi/2, for the tangent; or plus or minus pi, plus or minus 2pi, for the cotangent) in math routine *routine-name*.
CEE1V1	2	2017	The absolute value of the parameter was greater than or equal to *limit* in math routine *routine-name*.

result **(output)**
The result of the cotangent routine. The output range is:

$|result| \leq \Omega$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

The output range is:

$result \leq \Omega$

CEESxDIM—Positive difference

CEESxDIM returns the positive difference between two numbers by using one of the following equations:

If	Result
parm1 > *parm2*	Then, *result* = *parm1* - *parm2*
parm1 [*parm2*	Then, *result* = 0

The following routines are provided for the various data types supported:

CEESIDIM
32-bit binary integer

CEESSDIM
32-bit single floating-point number

CEESDDIM
64-bit double floating-point number

CEESQDIM
128-bit extended floating-point number

Syntax

```
►►──CEESxDIM──(──parm1──,──parm2──,──fc──,──result──)──────────────────────►◄
```

parm1 **(input)**
The first input to the positive difference routine. The input range is not restricted.

parm2 **(input)**
The second parameter to the positive difference routine. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the positive difference routine. The output range is the non-negative numbers.

$\leq \Omega$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxDVD—Floating-point complex divide

CEESxDVD performs the mathematical function of floating-point complex divide by using the equation:

$$result = \frac{parm1}{parm2}$$

The following routines are provided for the various data types supported:

CEESTDVD
32-bit single floating-point complex number

CEESEDVD
64-bit double floating-point complex number

CEESRDVD
128-bit extended floating-point complex number

Syntax

```
►►—CEESxDVD—(—parm1—,—parm2—,—fc—,—result—)——————————————————►◄
```

parm1 **(input)**
The first input to the math service. Any representable complex number can be used as input.

parm2 **(input)**
The second parameter to the math service. Do not set *parm2* to 0.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result (output)
The result of the floating-point complex divide routine.

CEESxERC—Error function complement

CEESxERC calculates the error function complement by using the equation:

$$result = \left(1 - \left(\frac{2}{\sqrt{\Pi}} \int_{o}^{parm1} e^{-u^2} du \right) \right)$$

The following routines are provided for the various data types supported:

CEESSERC
32-bit single floating-point number

CEESDERC
64-bit double floating-point number

CEESQERC
128-bit extended floating-point number

Syntax

```
►►—CEESxERC—(—parm1—,—fc—,—result—)——————————————————————————►◄
```

parm1 **(input)**
The input to the error function complement routine. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the error function complement routine. The output range is: $0 < result < 2$.

CEESxERF—Error function

CEESxERF calculates the error function by using the equation:

$$result = \frac{2}{\sqrt{\Pi}} \int_{o}^{parm1} e^{-u^2} du$$

The following routines are provided for the various data types supported:

CEESSERF
32-bit single floating-point number

CEESDERF
64-bit double floating-point number

CEESQERF
128-bit extended floating-point number

Syntax

```
▶▶──CEESxERF──(──parm1──,──fc──,──result──)──────────────────────▶◀
```

parm1 **(input)**
The input to the error function. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the error function. The output range is:

$|result| \leq 1$

CEESxEXP—Exponential base e

CEESxEXP calculates the mathematical function of e raised to a power by using the equation:

$result = e^{parm1}$

The following routines are provided for the various data types supported:

CEESSEXP
32-bit single floating-point number

CEESDEXP
64-bit double floating-point number

CEESQEXP
128-bit extended floating-point number

CEESTEXP
32-bit single floating-point complex number

CEESEEXP
64-bit double floating-point complex number

CEESREXP
128-bit extended floating-point complex number

Syntax

```
►►──CEESxEXP──(──parm1──,──fc──,──result──)─────────────────────────►◄
```

parm1 **(input)**
The input to the exponential base e routine. The input range varies, depending on the type of function and number:

For	Input range
Functions of real variables:	$\|parm1\| \leq 174.673$
The real part of complex numbers:	$\|\mathrm{Re}\ (parm1)\| < 174.673$
The imaginary part: single floating-point complex numbers:	$\|\mathrm{Im}\ (parm1)\| < 2^{18}\Pi$
The imaginary part: double floating-point complex numbers:	$\|\mathrm{Im}\ (parm1)\| < 2^{50}\ \Pi$

For	Input range
The imaginary part: extended floating-point complex numbers:	$\lvert \mathrm{Im}\,(parm1)\rvert < 2^{100}$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UP	2	2009	The value of the real part of the parameter was greater than *limit* in math routine *routine-name*.
CEE1UR	2	2011	The parameter was greater than *limit* in math routine *routine-name*.
CEE1UT	2	2013	The absolute value of the imaginary part of the parameter was greater than *limit* in math routine *routine-name*.
CEE1UV	2	2015	The absolute value of the imaginary part of the parameter was greater than or equal to *limit* in math routine *routine-name*.
CEE1V9	1	2025	An underflow occurred in math routine *routine-name*.

result **(output)**
The result of the exponential base e routine. The output range for functions of real variables is:

$$0 < \text{result} \le \Omega$$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxGMA—Gamma function

CEESxGMA performs the mathematical gamma function by using the equation:

$$result = \int_{0}^{\infty} u_{parm1} \text{-}1 e_{-U}\, du$$

The following routines are provided for the various data types supported:

CEESSGMA
32-bit single floating-point number

CEESDGMA
: 64-bit double floating-point number

Syntax

```
►►──CEESxGMA──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**

The input to the gamma function. The input range is:

$2^{-252} < parm1 < 57.5744$

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UL	2	2005	The value of the parameter was outside the valid range *range* in math routine *routine-name*.

result **(output)**

The result of the gamma function. The output range varies, depending on the type of routine:

For	Output range
Single-precision routines:	$0.88560 \le \text{result} \le \Omega$ $\Omega = 16^{63}(1\text{-}16^{6})$
Double-precision routines:	$\Omega = 16^{63}(1\text{-}16^{-14})$

CEESxIMG—Imaginary part of complex

CEESxIMG returns the imaginary part of a complex number using the equation *result* = *v*, where *parm1* = *u* + *vi*.

The following routines are provided for the various data types supported:

CEESTIMG
: 32-bit single floating-point complex number

CEESEIMG
: 64-bit double floating-point complex number

CEESRIMG
: 128-bit extended floating-point complex number

Syntax

```
►►──CEESxIMG──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**
The input to the math service. Any complex number can be used as input.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the math service is:

$|result| \le \Omega$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxINT—Truncation

CEESxINT returns the truncated value of the parameter by using the equation, where *m* is the greatest integer satisfying the relationship:

$result = (\text{sign of } parm1) \bullet n$, where $n = |parm1|$

$|parm1| = |m|$

The result is expressed as a floating-point number:

$|m| \le |parm1|$

The following routines are provided for the various data types supported:

CEESSINT
32-bit single floating-point number

CEESDINT
64-bit double floating-point number

CEESQINT
128-bit extended floating-point number

Syntax

```
►►──CEESxINT──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**
The input to the truncation routine. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the truncation routine. The output range is:

$|result| \leq \Omega$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxLGM—Log gamma

CEESxLGM performs the mathematical function of log gamma by using *either* of the following equations:

First equation	Alternate equation
$result = \log_e \Gamma(parm1)$	$result = \log_e \int_0^{\infty} u^{parm1} -1 e^{-U} \, du$

The following routines are provided for the various data types supported:

CEESSLGM
32-bit single floating-point number

CEESDLGM
64-bit double floating-point number

Syntax

```
►►──CEESxLGM──(──parm1──,──fc──,──result──)─────────────────────────►◄
```

parm1 **(input)**
The input to the log gamma routine. The input range is:

$$parm1 < 4.2913 \bullet 10^{73}$$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UL	2	2005	The value of the parameter was outside the valid range *range* in math routine *routine-name*.

result **(output)**
The result of the log gamma routine. The output range varies:

For	Output range
Single-precision routines:	$-0.12149 \leq result \leq \Omega$
Double-precision routines:	$\Omega = 16^{63}(1\text{-}16^{6})$ $\Omega = 16^{63}(1\text{-}16^{-14})$

CEESxLG1—Logarithm base 10

CEESxLG1 returns the logarithm base 10 of the input parameter by using the equation:

$$result = \log_{10} parm1$$

The following routines are provided for the various data types supported:

CEESSLG1
32-bit single floating-point number

CEESDLG1
64-bit double floating-point number

CEESQLG1
128-bit extended floating-point number

Syntax

```
►►──CEESxLG1──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**
The input to the log base 10 routine. The input range is: *parm1* > 0.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1US	2	2012	The parameter was less than or equal to *limit* in math routine *routine-name*.

result **(output)**
The result of the log base 10 routine. The output range is: *result* is greater than or equal to -78.268 and is less than or equal to 75.859.

CEESxLG2—Logarithm base 2

CEESxLG2 performs the mathematical function logarithm base 2 by using the equation:

$$result = \log_2 parm1$$

The following routines are provided for the various data types supported:

CEESSLG2
32-bit single floating-point number

CEESDLG2
64-bit double floating-point number

CEESQLG2
128-bit extended floating-point number

Syntax

```
►►──CEESxLG2──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**
The input to the log base 2 routine. The input range is: *parm1* > 0.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1US	2	2012	The parameter was less than or equal to *limit* in math routine *routine-name*.

result **(output)**
The result of the log base 2 routine. The output range is: *result* is greater than or equal to -260 and is less than or equal to 252.

CEESxLOG—Logarithm base e

CEESxLOG performs the mathematical function logarithm base e by using the equation:

$$result = \log_e parm1 \; (result = \ln parm1)$$

The following routines are provided for the various data types supported:

CEESSLOG
32-bit single floating-point number

CEESDLOG
64-bit double floating-point number

CEESQLOG
128-bit extended floating-point number

CEESTLOG
32-bit single floating-point complex number

CEESELOG
64-bit double floating-point complex number

CEESRLOG
128-bit extended floating-point complex number

Syntax

```
►►──CEESxLOG──(──parm1──,──fc──,──result──)──────────────────────►◄
```

parm1 **(input)**
The input to the log base e routine. The input range varies:

For	Input range
Real numbers:	*parm1* > 0

For	Input range
Complex numbers:	*parm1* is not equal to 0

G

`fc` **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1US	2	2012	The parameter was less than or equal to *limit* in math routine *routine-name*.
CEE1V2	2	2018	The real and imaginary parts of the parameter were equal to *limit* in math routine *routine-name*.

`result` **(output)**
The result of the log base e routine. The output range for functions of real variables is: *result* is greater than or equal to -180.218 and is less than or equal to ≤ 174.673.

CEESxMLT—Floating-point complex multiply

CEESxMLT performs the mathematical function floating-point complex multiply by using the equation:

$$result = parm1 \bullet parm2$$

The following routines are provided for the various data types supported:

CEESTMLT
32-bit single floating-point complex number

CEESEMLT
64-bit double floating-point complex number

CEESRMLT
128-bit extended floating-point complex number

Syntax

```
►►—CEESxMLT—(—parm1—,—parm2—,—fc—,—result—)——————————————————►◄
```

`parm1` **(input)**
The first input to the math service. Any representable complex number can be used as input.

parm2 **(input)**
The second parameter to the math service. Any representable complex number can be used as input.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the floating-point complex multiply routine.

CEESxMOD—Modular arithmetic

CEESxMOD performs the mathematical function modular arithmetic by using the equation:

result = *parm*1(modulo *parm*2)

The expression *parm1*(modulo *parm2*) is defined as follows, with the brackets indicating an integer part:

*parm*1 - [(*parm*1/*parm*2)• *parm*2]

That is, the largest integer whose magnitude does not exceed the magnitude of *parm1*/*parm2* is used.

The sign of the integer is the same as the sign of

*parm*1• *parm*2

The following routines are provided for the various data types supported:

CEESIMOD
32-bit binary integer

CEESSMOD
32-bit single floating-point number

CEESDMOD
64-bit double floating-point number

CEESQMOD
128-bit extended floating-point number

Syntax

```
►►──CEESxMOD──(──parm1──,──parm2──,──fc──,──result──)──────────────────►◄
```

parm1 **(input)**
The first parameter to the modular arithmetic routine. The input range is not restricted.

parm2 **(input)**
The second parameter to the modular arithmetic routine. The input range is: *parm2* is not equal to 0.

If *parm2* = 0, the modulus routine is undefined. In addition, a divide exception is recognized and an interrupt occurs.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the mod routine. The output range is:

$|result| \leq \Omega$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxNIN—Nearest integer

CEESxNIN performs the mathematical function nearest integer by using the equation:

$result$ = (sign of $parm1$) • n

If $parm1$ m 0, then $n = [|parm1 + .5|]$
If $parm1 < 0$, then $n = [|parm1 - .5|]$

$n = |m|$, where m is the greatest integer satisfying the relationship $|m|$ [$|parm1 + .5|$, or $|m|$ [$|parm1 - .5|$, respectively.

The following routines are provided for the various data types supported:

CEESSNIN
32-bit single floating-point number

CEESDNIN
64-bit double floating-point number

Syntax

```
►►──CEESxNIN──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**
The input to the nearest integer routine. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the nearest integer routine. The output parameter is an unrestricted type I, a 32-bit binary integer.

CEESxNWN—Nearest whole number

CEESxNWN performs the mathematical function nearest whole number by using the equation:

result = (sign of parm1)•v

If *parm*1 $\geq$ 0, then *v* = [|*parm*1 +.5|]. If *parm*1 <0, then *v* = [|*parm*1 - .5|].

v = |*m*|, where *m* is the greatest integer satisfying the relationship |*m*| $\leq$ |*parm*1 + .5|, or |*m*| $\leq$ |*parm*1 - .5|, respectively.

and the resulting *v* is expressed as a floating-point number.

The following routines are provided for the various data types supported:

CEESSNWN
32-bit single floating-point number

CEESDNWN
64-bit double floating-point number

Syntax

```
▶▶──CEESxNWN──(──parm1──,──fc──,──result──)──────────────────────▶◀
```

parm1 **(input)**
The input to the nearest whole number routine. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the nearest whole number routine. The output range varies:

For	Output range
Single-precision routines:	$\lvert result\rvert \leq \Omega$ $\Omega = 16^{63}(1\text{-}16^{6})$
Double-precision routines:	$\Omega = 16^{63}(1\text{-}16^{-14})$

CEESxSGN—Transfer of sign

CEESxSGN performs the mathematical function transfer of sign by using *either* of the two equations:

$$result = \lvert parm1\rvert \text{ if } parm2 \geq 0 \text{ or } result = -\lvert parm1\rvert \text{ if } parm2 < 0.$$

The following routines are provided for the various data types supported:

CEESISGN
32-bit binary integer

CEESSSGN
32-bit single floating-point number

CEESDSGN
64-bit double floating-point number

CEESQSGN
128-bit extended floating-point number

Syntax

```
►►—CEESxSGN—(—parm1—,—parm2—,—fc—,—result—)——————————————————►◄
```

parm1 **(input)**
The first input to the transfer of sign routine. The input range is not restricted.

parm2 **(input)**
The second parameter to the transfer of sign routine. The input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the transfer of sign routine. The output range is:

$|result| \leq \Omega$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxSIN—Sine

CEESxSIN returns the sine of the parameter by using the equation: *result* = sin (*parm1*)

The following routines are provided for the various data types supported:

CEESSSIN
32-bit single floating-point number

CEESDSIN
64-bit double floating-point number

CEESQSIN
128-bit extended floating-point number

CEESTSIN
32-bit single floating-point complex number

CEESESIN
64-bit double floating-point complex number

CEESRSIN
128-bit extended floating-point complex number

Syntax

```
►►──CEESxSIN──(──parm1──,──fc──,──result──)──────────────────────────────►◄
```

parm1 **(input)**

The input, in radians, to the sine routine. For real functions, the input range varies, depending on the value of *parm1*:

If	Input range
parm1 is a 32-bit single floating-point number:	$\|parm1\| < (2^{18} \bullet \Pi)$
parm1 is a 64-bit double floating-point number:	$\|parm1\| < (2^{50} \bullet \Pi)$
parm1 is an extended floating-point number:	$\|parm1\| < 2^{100}$

For complex functions, the input range differs for the imaginary and real parts of the input.

Part	Input range
For the imaginary part:	$\| \mathrm{Im}\ (parm1)\| < 174.673$
For the real part: single floating-point complex numbers	$\| \mathrm{Re}\ (parm1)\| < (2^{18} \bullet \Pi)$
For the real part: double floating-point complex numbers	$\| \mathrm{Re}\ (parm1)\| < (2^{50} \bullet \Pi)$
For the real part: extended floating-point complex numbers	$\| \mathrm{Re}\ (parm1)\| < 2^{100}$

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UT	2	2013	The absolute value of the imaginary part of the parameter was greater than *limit* in math routine *routine-name*.
CEE1V1	2	2017	The absolute value of the parameter was greater than or equal to *limit* in math routine *routine-name*.
CEE1V3	2	2019	The absolute value of the real part of the parameter was greater than or equal to *limit* in math routine *routine-name*.

`result` **(output)**
The result of the sine routine. The output range for functions of real variables is: *result* is greater than or equal to -1 and is less than or equal to 1.

CEESxSNH—Hyperbolic sine

CEESxSNH performs the mathematical function hyperbolic sine by using the equation: *result* = sinh (*parm1*)

The following routines are provided for the various data types supported:

CEESSSNH
32-bit single floating-point number

CEESDSNH
64-bit double floating-point number

CEESQSNH
128-bit extended floating-point number

CEESTSNH
32-bit single floating-point complex number

CEESESNH
64-bit double floating-point complex number

CEESRSNH
128-bit extended floating-point complex number

Syntax

```
►►──CEESxSNH──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

`parm1` **(input)**
The input to the hyperbolic sine routine. The input range varies.

For	Input range
Input range for reals:	$|parm1| < 175.366$

For complex functions, the input range differs for the imaginary and real parts of the input. The input range of the imaginary part also differs depending on the precision of *parm1*.

For	Input range
Complex functions: real part	$|Re(parm1)| < 174.673$
Imaginary part: single floating-point complex numbers	$|Im(parm1)| < 2^{18} \bullet \Pi$
Imaginary part: double floating-point complex numbers	$|Im(parm1)| < 2^{50} \bullet \Pi$
Imaginary part: extended floating-point complex numbers:	$|Im(parm1)| < 2^{100}$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the parameter was greater than *limit* in math routine *routine-name*.

result **(output)**
The result of the hyperbolic sine routine. The output range for functions of real variables is:

$$|result| \leq \Omega$$

CEESxSQT—Square root

CEESxSQT returns the square root of *parm1* by using the equation:

$$result = \sqrt{parm1}$$

The following routines are provided for the various data types supported:

CEESSSQT
32-bit single floating-point number

CEESDSQT
64-bit double floating-point number

CEESQSQT
128-bit extended floating-point number

CEESTSQT
32-bit single floating-point complex number

CEESESQT
64-bit double floating-point complex number

CEESRSQT
128-bit extended floating-point complex number

Syntax

```
►►──CEESxSQT──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**
The input to the square root routine. The input range for real number functions is: *parm1* is greater than, or equal to, 0.

For complex numbers, the input range is not restricted.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UQ	2	2010	The parameter was less than *limit* in math routine *routine-name*.

result **(output)**
The result of the square root routine. The output range for functions of real variables is:

$0 \le result \le \Omega^{1/2}$

For single-precision routines, $\Omega = 16^{63}(1 - 16^{-6})$
For double-precision routines, $\Omega = 16^{63}(1 - 16^{-14})$
For extended-precision routines, $\Omega = 16^{63}(1 - 16^{-28})$

CEESxTAN—Tangent

CEESxTAN returns the tangent of the parameter by using the equation: *result* = tan (*parm1*)

The following routines are provided for the various data types supported:

CEESSTAN
32-bit single floating-point number

CEESDTAN
64-bit double floating-point number

CEESQTAN
128-bit extended floating-point number

CEESTTAN
32-bit single floating-point complex number

CEESETAN
64-bit double floating-point complex number

CEESRTAN
128-bit extended floating-point complex number

Syntax

```
►►──CEESxTAN──(──parm1──,──fc──,──result──)──────────────────────────►◄
```

parm1 **(input)**

The input, in radians, to the tangent routine. The input range varies, depending on the value of *parm1*:

For	Input range
parm1 is a single floating-point number:	$\|parm1\| < 2^{18} \bullet \Pi$
parm1 is a double floating-point number:	$\|parm1\| < 2^{50} \bullet \Pi$
parm1 is an extended floating-point number:	< $\|parm1\| < 2^{100}$

Also, for extended and as complex functions, this argument cannot approach odd multiples of:

$\Pi/2$

fc **(output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to “Invoking callable services” on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UI	2	2002	The parameter value was too close to one of the singularities (plus or minus pi/2, plus or minus 3pi/2, for the tangent; or plus or minus pi, plus or minus 2pi, for the cotangent) in math routine *routine-name*.
CEE1V1	2	2017	The absolute value of the parameter was greater than or equal to *limit* in math routine *routine-name*.
CEE1V9	1	2025	An underflow occurred in math routine *routine-name*.

result **(output)**

The result of the tangent routine. The output range for functions of real variables is:

$|result| \leq \Omega$

CEESxTNH—Hyperbolic tangent

CEESxTNH performs the mathematical function hyperbolic tangent by using the equation: *result* = tanh (*parm1*)

The following routines are provided for the various data types supported:

CEESSTNH
32-bit single floating-point number

CEESDTNH
64-bit double floating-point number

CEESQTNH
128-bit extended floating-point number

CEESTTNH
32-bit single floating-point complex number

CEESETNH
64-bit double floating-point complex number

CEESRTNH
128-bit extended floating-point complex number

Syntax

```
►►—CEESxTNH—(—parm1—,—fc—,—result—)——————————————————————————►◄
```

parm1 **(input)**
The input to the hyperbolic tangent routine. The input range is not restricted for real functions. For complex functions, *parm1* must not approach odd multiples of:

$\Pi/2 \bullet i$, where $i = \sqrt{-1}$

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.

result **(output)**
The result of the hyperbolic tangent routine. The output range for functions of real variables is:

$|result| < 1$

CEESxXPx—Exponentiation

CEESxXPx performs the mathematical function exponentiation by using the equation:

$result = parm1^{parm2}$

The following routines are provided for the various data types supported:

CEESIXPI
32-bit binary integer raised to a 32-bit binary integer

CEESSXPI
32-bit single floating-point number raised to a 32-bit binary integer

CEESDXPI
64-bit double floating-point number raised to a 32-bit binary integer

CEESQXPI
128-bit extended floating-point number raised to a 32-bit binary integer

CEESTXPI
32-bit single floating-point complex number raised to a 32-bit binary integer

CEESEXPI
64-bit double floating-point complex number raised to a 32-bit binary integer

CEESRXPI
128-bit extended floating-point complex number raised to a 32-bit binary integer

CEESSXPS
32-bit single floating-pointing point raised to a 32-bit single floating-point

CEESDXPD
64-bit double floating-point raised to a 64-bit double floating-point

CEESQXPQ
128-bit extended floating-point raised to a 128-bit extended floating-point

CEESTXPT
32-bit single floating-point complex raised to a 32-bit single floating-point complex

CEESEXPE
64-bit double floating-point complex raised to a 64-bit double floating-point complex

CEESRXPR
128-bit extended floating-point complex raised to a 128-bit extended floating-point complex

Syntax

```
►►──CEESxXPx──(──parm1──,──parm2──,──fc──,──result──)──────────────────────►◄
```

parm1 **(input)**
The input for the base of the exponentiation routine. The input range varies.

For functions of real variables:	*if* → *parm1* = 0, then *parm2* > 0.
If *parm1* is a 32-bit number and *parm1* < 0:	$|parm1| \leq (16^{6} - 1)$ and *parm2* = *'awholenumber'*
If *parm1* is a 64-bit number and *parm1* < 0:	$|parm1| \leq (16^{14} - 1)$ and *parm2* = *'awholenumber'*
If *parm1* is a 128-bit number and *parm1* < 0:	$|Parm1| \leq (16^{28} - 1)$ and *parm2* = *'awholenumber'*

The input range for functions of complex variables: If Re(*parm1*) = 0 and Im(*parm1*) = 0, then Re(*parm2*) must be positive.

parm2 **(input)**
The input for the power of the exponentiation routine. The type is determined by the eighth character of the service name.

fc **(output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. If you choose to omit this parameter, refer to "Invoking callable services" on page 114 for the appropriate syntax to indicate that the feedback code was omitted.

The following symbolic conditions can result from this service:

Code	Severity	Message number	Message text
CEE000	0	—	The service completed successfully.
CEE1UJ	2	2003	For an exponentiation operation (I**J) where I and J are integers, I was equal to zero and J was less than or equal to zero in math routine *routine-name*.
CEE1UK	2	2004	For an exponentiation operation (R**I) where R is real and I is an integer, R was equal to zero and I was less than or equal to zero in math routine *routine-name*.
CEE1UL	2	2005	The value of the parameter was outside the valid range *range* in math routine *routine-name*.
CEE1UM	2	2006	For an exponentiation operation (R**S) where R and S are real values, R was equal to zero and S was less than or equal to zero in math routine *routine-name*.
CEE1UN	2	2007	The exponent exceeded *limit* in math routine *routine-name*.

Code	Severity	Message number	Message text
CEE1UO	2	2008	For an exponentiation operation (Z**P) where the complex base Z equals zero, the real part of the complex exponent P, or the integer exponent P was less than or equal to zero in math routine *routine-name*.
CEE1V4	2	2020	For an exponentiation operation (R**S) where R and S are real values, either R is equal to zero and S is negative, or R is negative and S is not an integer whose absolute value is less than or equal to *limit* in math routine *routine-name*.
CEE1V5	2	2021	For an exponentiation operation (X**Y), the parameter combination of Y*log2(X) generated a number greater than or equal to *limit* in math routine *routine-name*.
CEE1V8	2	2024	An overflow occurred in math operation (X**Y).
CEE1V9	1	2025	An underflow occurred in math routine *routine-name*. The output value from the math routine is undefined.
CEE1VF	2	2031	The value of the argument was a non-positive whole number in math routine (X**Y).

result **(output)**
The result of the exponentiation routine. The output range for functions of real variables is:

$|result| \leq \Omega$

Examples of math services

The following sections contain examples of calls to various math services made from supported languages.

C/C++ math service examples

Table 29 on page 476 shows code examples of calling various math services from C/C++.

Examples

Table 29. C/C++ examples

Function called	Code example

Log base 10 and modular arithmetic (CEESDGL1 and CEESIMOD)	`#include <leawi.h>`
	`#include <string.h>`
	`#include <stdio.h>`
	
	`int main (void) {`
	
	`  _FLOAT8 f1,result;`
	`  _INT4 int1, int2, intr;`
	
	`  _FEEDBACK fc;`
	`  #define SUCCESS "\0\0\0\0"`
	
	`  f1 = 1000.0;`
	
	`  CEESDLG1(&f1,&fc,&result);`
	
	`  if (memcmp(&fc,SUCCESS,4) != 0) {`
	`     printf("CEESDLG1 failed with message number %d\n",`
	`            fc.tok_msgno);`
	`     exit(2999);`
	`  }`
	
	`  printf("%f log base 10 is %f\n",f1,result);`
	
	`  int1 = 39;`
	`  int2 = 7;`
	`  CEESIMOD(&int1,&int2,&fc,&intr);`
	
	`  if (memcmp(&fc,SUCCESS,4) != 0) {`
	`     printf("CEESIMOD failed with message number %d\n",`
	`            fc.tok_msgno);`
	`     exit(2999);`
	`  }`
	
	`  printf("%d modulo %d is %d\n",int1,int2,intr);`
	`}`

COBOL math service examples

Table 30 shows code examples of calling various math services from COBOL.

Table 30. COBOL examples

Function called	Code example

Log base e (CEESSLOG)	`77  ARG1RS  COMP-1.`
	`77  FBCODE  PIC X(12).`
	`77  RESLTRS COMP-1.`
	
	`    CALL "CEESSLOG" USING ARG1RS , FBCODE ,`
	`    RESLTRS.`

Log base 10 (CEESDLG1)	`77  ARG1RL  COMP-2.`
	`77  FBCODE  PIC X(12).`
	`77  RESLTRL COMP-2.`
	
	`    CALL "CEESDLG1" USING ARG1RL , FBCODE ,`
	`    RESLTRL.`

Table 30. COBOL examples (continued)

Function called	Code example

Exponentiation (CEESIXPI)	
	```
	77   ARG1IS  PIC S9(9) COMP.
	77   ARG2IS  PIC S9(9) COMP.
	77   FBCODE  PIC X(12).
	77   RESLTIS PIC S9(9) COMP.

	    CALL "CEESIXPI" USING ARG1IS , ARG2IS ,
	    FBCODE , RESLTIS.
	```

Exponentiation (CEESSXPI)	
	```
	77   ARG1RS  COMP-1.
	77   ARG2IS  PIC S9(9) COMP.
	77   FBCODE  PIC X(12).
	77   RESLTRS COMP-1.

	    CALL "CEESSXPI" USING ARG1RS , ARG2IS ,
	    FBCODE , RESLTRS.
	```

Arctangent2 (CEESSAT2)	
	```
	77   ARG1RS  COMP-1.
	77   ARG2RS  COMP-1.
	77   FBCODE  PIC X(12).
	77   RESLTRS COMP-1.

	     CALL "CEESSAT2" USING ARG1RS , ARG2RS ,
	     FBCODE , RESLTRS.
	```

PL/I math service examples

Table 31 shows code examples of calling various math services from PL/I.

Table 31. PL/I examples

Function called	Code example
Modular arithmetic and log base e (CEESIMOD and CEESSLOG)	see below

```
PLIMATH: PROC OPTIONS(MAIN);

   DCL CEESSLOG ENTRY OPTIONS(ASM) EXTERNAL;
   DCL CEESIMOD ENTRY OPTIONS(ASM) EXTERNAL;

   DCL ARG1  RESULT REAL FLOAT DEC (6);
   DCL ARGM1 ARGM2  RES2 FLOAT BINARY(21)
   DCL FC    CHARACTER (12);

  /* Call log base e routine, which has    */
  /* only one input parameter              */
   CALL CEESSLOG (ARG1, FC, RESULT)

   IF ( FC = '000000000000000000000000'X )
     THEN DO;
       PUT SKIP LIST
          ( 'Error occurred in call to CEESSLOG.' );
     ELSE;

  /* Call modular arithmetic routine,      */
  /* which has two input parameters        */
   CALL CEESIMOD (ARGM1, ARGM2, FC, RES2);

   IF ( FC = '000000000000000000000000'X )
     THEN DO;
       PUT SKIP LIST
          ( 'Error occurred in call to CEESIMOD.' );
     ELSE;
   END;
```

Table 31. PL/I examples (continued)

Function called	Code example

Double-precision complex tangent (CEESETAN)	`TRYETAN:  PROCEDURE OPTIONS( MAIN );`
	
	`   DECLARE FC          CHARACTER(12);`
	`   DECLARE PARM1       COMPLEX FLOAT BINARY(53);`
	`   DECLARE RESULT      COMPLEX FLOAT BINARY(53);`
	
	`   DECLARE CEESETAN    ENTRY(COMPLEX FLOAT BINARY(53),`
	`                          *, COMPLEX FLOAT BINARY(53))`
	`                          OPTIONS(ASSEMBLER) EXTERNAL;`
	
	`   PARM1 = COMPLEX(7,1.1);`
	`   CALL CEESETAN ( PARM1, FC, RESULT);`
	`   IF ( FC ¬= '000000000000000000000000'X) THEN`
	`         PUT SKIP LIST( 'Error in call to CEESETAN.');`
	`   ELSE`
	`         PUT SKIP LIST( 'Result is ' || RESULT);`
	`END TRYETAN;`

Part 3. Appendixes

Appendix A. IBM-supplied country code defaults

Table 32 contains the currency symbols and default picture strings for the *country_code* parameters of the COUNTRY runtime option and the national language support callable services. See "COUNTRY" on page 22 and the services listed in Table 17 on page 112 for more information.

Note: In the table, some currency symbols are shown as hexadecimal strings. How these are displayed depends on the `codeset` in use by the terminal device. For example, when using code page 01140, X'9F404040' is displayed as the Euro symbol followed by three blanks.

Table 32. Defaults currency and picture strings based on COUNTRY setting

Country/ region	Country code	Decimal separator	Thousand separator	Currency symbol	Int. currency symbol	Time picture string	Date picture string	Date and time picture string
Afghanistan	AF	,	.	X'9F404040'	AFN	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Albania	AL	,	.	Lek	ALL	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Algeria	DZ	,	.		DZD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Andorra	AD	,	.	X'9F404040'	EUR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Angola	AO	,	.	X'9F404040'	AOA	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Antigua and Barbuda	AG	,	.	X'9F404040'	XCD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Argentina	AR	,	.	A	ARS	HH.MI.SS	DD/MM/YY	DD/MM/YY HH.MI.SS
Armenia	AM	`	:	AMD	dr:	HH:MI:SS'	MM/DD/YY	Mmmmmmmmmz DD, YYYY HH:MI:SS
Australia	AU	.	,	$	AUD	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Austria	AT	,	.	X'9F404040'	EUR	HH:MI:SS,999	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS,999
Azerbaijan	AZ	.	,	AZN	man	HH:MI:SS	MM/DD/YY	ZD Mmmmmmmmz , YY HH.MI.SS'
Bahamas	BS	,	.	X'9F404040'	BSD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bahrain	BH	,	.		BHD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Bangladesh	BD	,	.	X'9F404040'	BDT	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Barbados	BB	,	.	X'9F404040'	BBD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Belgium	BE	,	.	X'9F404040'	EUR	HH:MI:SS,999	DD/MM/YY	DD/MM/YY HH:MI:SS,999
Benin	BJ	,	.	X'9F404040'	XOF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bermuda	BM	,	.	X'9F404040'	BMD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bolivia	BO	,	.	BS	BOB	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Bosnia/ Herzegovina	BA	,	.	Din	BAM	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Botswana	BW	,	.	X'9F404040'	BWP	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Brazil	BR	,	.	NCz$	BRL	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Brunei Darussalam	BN	,	.	X'9F404040'	BND	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bulgaria	BG	,	.	Lv	BGN	HH:MI:SS	YYYY-RRRZ-DD	YYYY-RRRZ-DD HH:MI:SS
Burkina Faso (Upper Volta)	BF	,	.	X'9F404040'	XOF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Burma	BU	,	.	X'9F404040'	MMK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Canada	CA	.	,	$	CAD	HH:MI:SS.99	YY-MM-DD	YY-MM-DD HH:MI:SS.99
Cayman Islands	KY	,	.	X'9F404040'	KYD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Chad	TD	,	.	X'9F404040'	XAF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 32. Defaults currency and picture strings based on COUNTRY setting (continued)

Country/ region	Country code	Decimal separator	Thousand separator	Currency symbol	Int. currency symbol	Time picture string	Date picture string	Date and time picture string
Chile	CL	,	.	$	CLP	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Colombia	CO	,	.	$	COP	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Costa Rica	CR	,	.	c/	CRC	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Croatia	HR	,	.	Din	HRK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Cuba	CU	,	.	X'9F404040'	CUP	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Cyprus	CY	,	.	X'9F404040'	CYP	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Czech Republic	CZ	,	.	X'D247A240'	CZK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Denmark	DK	,	.	kr	DKK	HH.MI.SS,99	DD-MM-YY	DD-MM-YY HH.MI.SS,99
Dominican Republic	DO	.	,	$	DOP	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Ecuador	EC	,	.	$	USD	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Egypt	EG	,	.		EGP	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
El Salvador	SV	.	,	c/.	SVC	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Estonia	EE	,		Kr	EEK	HH:MI:SS	DD-MM-YYYY	DD-MM-YYYY HH:MI:SS
Ethiopia	ET	,	.	X'9F404040'	ETB	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Finland	FI	,		X'5A404040'	EUR	HH.MI.SS,999	DD.MM.YYYY	DD.MM.YYYY HH.MI.SS,99
France	FR	,		X'9F404040'	EUR	HH:MI:SS,9	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS,9
Gabon	GA	,	.	X'9F404040'	XAF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Gambia	GM	,	.	X'9F404040'	GMD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Georgia, Republic of	GE	:	,	GEL	GEL	HH:MI:SS	YYYY-MM-DD	YYYY Mmz ZD HH:MI:SS
Germany	DE	,	.	X'9F404040'	EUR	HH:MI:SS	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS
Ghana	GH	,	.	X'9F404040'	GHC	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Greece	GR	,	.	X'FC404040'	EUR	HH:MI:SS.999	DD/MM/YY	DD/MM/YY HH:MI:SS.999
Guatemala	GT	.	,	Q	GTQ	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Guinea-Bissau	GW	,	.	X'9F404040'	XOF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Guyana	GY	,	.	X'9F404040'	GYD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Haiti	HT	,	.	X'9F404040'	HTG	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Honduras	HN	.	,	L.	HNL	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
China (Hong Kong S.A.R.)	HK	,	.	X'9F404040'	HKD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Hungary	HU	,		FT	HUF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Iceland	IS	,	.	kr	ISK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
India	IN	,	.	X'9F404040'	INR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Indonesia	ID	,	.	X'9F404040'	IDR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Iran	IR	,	.		IRR	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Iraq	IQ	,	.		IQD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Ireland	IE	,	,	X'9F404040'	EUR	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Israel	IL	.	,	NIS	ILS	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Italy	IT	,	.	X'9F404040'.	EUR	HH.MI.SS,999	DD/MM/YY	DD/MM/YY HH.MI.SS,999
Jamaica	JM	,	.	X'9F404040'	JMD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Japan	JP	.	,	X'5B404040'	JPY	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Jordan	JO	,	.		JOD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Kenya	KE	,	.	X'9F404040'	KES	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Korea, Republic of	KR	.	,	X'E0404040'	KRW	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Kuwait	KW	,	.		KWD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Latvia	LV	,		Ls	LVL	HH:MI:SS	YYYY.DD.RRRZ	YYYY.DD.RRRZ HH:MI:SS
Lebanon	LB	,	.		LBP	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS

Table 32. Defaults currency and picture strings based on COUNTRY setting (continued)

Country/ region	Country code	Decimal separator	Thousand separator	Currency symbol	Int. currency symbol	Time picture string	Date picture string	Date and time picture string
Lesotho	LS	,	.	X'9F404040'	ZAR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Liberia	LR	,	.	X'9F404040'	LRD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Libya	LY	,	.		LYD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Liecht- enstein	LI			X'X'9F404040' '	CHF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Lithuania	LT	,	.	Lt	LTL	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Luxembourg	LU	,	.	X'9F404040'	EUR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
China (Macau S.A.R.)	MO	,	.	X'9F404040'	MOP	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Macedonia, Former Yugoslav Republic of	MK	,	.	Den	MKD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Madagascar	MG	,	.	X'9F404040'	MGA	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malawi	MW	,	.	X'9F404040'	MWK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malaysia	MY	,	.	X'9F404040'	MYR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malta	MT	,	.	X'9F404040''	EUR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mali	ML	,	.	X'9F404040'	XOF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mauritania	MR	,	.	X'9F404040'	MRO	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mauritius	MU	,	.	X'9F404040'	MUR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mexico	MX	.	,	$	MXN	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Monaco	MC	,	.	X'9F404040'	EUR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Morocco	MA	,	.		MAD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Mozam- bique	MZ	,	.	X'9F404040'	MZN	HH:MI:SS	YYYY-MM-DD HH:MI:SS	
Namibia	NA	,	.	X'9F404040'	ZAR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Netherlands	NL	,	.	X'9F404040'	EUR	HH:MI:SS	DD-MM-YY	DD-MM-YY HH:MI:SS
Netherlands Antilles	AN	,	.	X'9F404040'	ANG	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
New Caledonia	NC	,	.	X'9F404040'	XPF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
New Zealand	NZ	.	,	$	NZD	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Nicaragua	NI	.	,	X'9F404040'	NIO	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Niger	NE	,	.	X'9F404040'	XOF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Nigeria	NG	,	.	X'9F404040'	NGN	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Norway	NO	,	.	kr	NOK	HH:MI:SS,999	DD.MM.YY	DD.MM.YY HH:MI:SS,999
Oman	OM	,	.		OMR	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Pakistan	PK	,	.		PKR	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Panama	PA	.	,	B/	PAB	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Papua New Guinea	PG	,	.	X'9F404040'	PGK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Paraguay	PY	,	.	Gs.	PYG	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
People's Republic of China	CN	.	,	X'5B404040'	CNY	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Peru	PE	.	,	I/.	PEN	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Philippines	PH	,	.	X'9F404040'	PHP	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Poland	PL	,	.	X'E99A4040'	PLN	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Portugal	PT	,	.	X'9F404040'.	EUR	HH:MI:SS	DD-MM-YYYY	DD-MM-YYYY HH:MI:SS
Puerto Rico	PR	.	,	$	USD	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Qatar	QA	,	.		QAR	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS

Table 32. Defaults currency and picture strings based on COUNTRY setting (continued)

Country/ region	Country code	Decimal separator	Thousand separator	Currency symbol	Int. currency symbol	Time picture string	Date picture string	Date and time picture string
Taiwan	TW	.	,	$	TWD	HH:MI:SS.999	YY/MM/DD	YY/MM/DD HH:MI:SS.999
Romania	RO	,	.	Lei	RON	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Russia	RU	,	.	Rub	RUB	HH:MI:SS	DD mmm. YYYY g.	DD mmm. YYYY g. HH:MI:SS
Saint Lucia	LC	,	.	X'9F404040'	XCD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Saudi Arabia	SA	,	.		SAR	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Senegal	SN	,	.	X'9F404040'	XOF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Seychelles	SC	,	.	X'9F404040'	SCR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sierra Leone	SL	,	.	X'9F404040'	SLL	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Singapore	SG	,	.	X'9F404040'	SGD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Slovakia	SK	,	.	X'D247A240'	SKK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Slovenia	SI	,	.	X'9F404040'	EUR	HH:MI:SS	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS
Somalia	SO	,	.	X'9F404040'	SOS	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
South Africa	ZA	.		R	ZAR	HHhMI:SS.999	YYYY-MM-DD	YYYY-MM-DD HHhMI:SS.999
Spain	ES	,	.	X'9F404040'	EUR	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Sri Lanka	LK	,	.	X'9F404040'	LKR	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sudan	SD	,	.		SDD	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Surinam	SR	,	.	X'9F404040'	SRD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Swaziland	SZ	,	.	X'9F404040'	SZL	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sweden	SE	,	.	kr	SEK	kl HH.MI.SS	YYYY-MM-DD	YYYY-MM-DD kl HH.MI.SS
Switzerland	CH	,	.	Fr	CHF	HH,MI,SS	DD. Mmmmmmmmz YYYY	DD. Mmmmmmmmz YYYY HH,MI,SS
Syria	SY	,	.		SYP	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Tanzania	TZ	,	.	X'9F404040'	TZS	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Thailand	TH	.	,	X'70404040'	THB	HH:MI:SS	DD/MM/YYYY	DD/MM/YYYY HH:MI:SS
Togo	TG	,	.	X'9F404040'	XOF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Tunisia	TN	,	.		TND	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Turkey	TR	,	.	YTL	TRY	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Uganda	UG	,	.	X'9F404040'	UGX	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Union of Soviet Socialist Republics (See note below)	SU	,	.	Rub	RUB	HH:MI:SS	DD mmm. YYYY g.	DD mmm. YYYY g. HH:MI:SS
United Arab Emirates	AE	,	.		AED	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
United Kingdom	GB	.	,	X'5B404040'	GBP	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
United States	US	.	,	$	USD	ZH:MI:SS AP	MM/DD/YY	MM/DD/YY ZH:MI:SS AP
Uruguay	UY	,	.	N$	UYU	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Vanuatu	VU	,	.	X'9F404040'	VUV	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Venezuela	VE	,	.	BsF	VEB	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Western Samoa	WS	,	.	X'9F404040'	WST	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Yemen	YE	,	.		YER	HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Yugoslavia	YU	,	.	Din	YUM	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zaire	ZR	,	.	X'9F404040'	CDF	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zambia	ZM	,	.	X'9F404040'	ZMK	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zimbabwe	ZW	,	.	X'9F404040'	ZWD	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 32. Defaults currency and picture strings based on COUNTRY setting (continued)

Country/ region	Country code	Decimal separator	Thousand separator	Currency symbol	Int. currency symbol	Time picture string	Date picture string	Date and time picture string

Note:

1. The Czechoslovakia country code CS is obsolete. Use either the Czech Republic country code CZ, or the Slovakia country code SK.
2. Country code DE was used for the former Federal Republic of Germany and now represents the settings for Germany. The DD country code, used in the past for the German Democratic Republic, is obsolete and should be replaced by DE.
3. The SU country code is obsolete. Use the following country codes for the appropriate country: Estonia, EE; Latvia, LV; Lithuania, LT; Russian Federation, RU.

Appendix B. Date and time services tables

This appendix contains information to help you use Language Environment date and time callable services. Included are tables for picture term and national language era usage.

Table 33. Picture character terms used in picture strings for date and time services

Picture terms	Explanations	Valid values	Note
Y	1-digit year	0-9	Y valid for output only.
YY	2-digit year	00-99	YY assumes range set by CEESCEN.
YYY	3-digit year	000-999	YYY and ZYY valid only if used with <JJJJ>, <CCCC>, or <CCCCCCCC>.
ZYY	3-digit year within era	1-999	
YYYY	4-digit year	1582-9999	
<JJJJ>	Japanese era name in DBCS characters	Heisei (X'0E458D45BA0F') Showa (X'0E45B3457A0F') Taisho (X'0E455B45770F') Meiji (X'0E45A645840F')	Affects YY field: if <JJJJ> specified, YY means the year within Japanese era, for example, 1988 equals Showa 63. See example in Table 34 on page 488.
<CCCC> <CCCCCCCC>	Era name in DBCS characters	(X'0E4D8256CE0F') (X'0E4C845ADD4D8256CE0F')	Affects YY field: if <CCCC> specified, YY means the year within the era. See example in Table 34 on page 488.
MM ZM	2-digit month 1- or 2-digit month	01-12 1-12	For output, leading zero suppressed. For input, ZM treated as MM.
RRRR RRRZ	Roman numeral month	Iƀƀƀ-XIIƀ (left-aligned)	For input, source string is folded to uppercase. For output, uppercase only. I=Jan, II=Feb, ..., XII=Dec.
MMM	3-char month, uppercase	JAN-DEC	For input, source string always folded to uppercase. For output, M generates uppercase and m generates lowercase. Output is padded with blanks (ƀ) (unless Z specified) or truncated to match the number of M's, up to 20.
Mmm	3-char month, mixed case	Jan-Dec	
Mmmm...m	3-20 char mo., mixed case	January-December	
MMMM...M	3-20 char mo., uppercase	JANUARY-DECEMBER	
MMMMMMMMMZ	trailing blanks suppressed	JANUARYƀƀ-DECEMBERƀ	
Mmmmmmmmmz	trailing blanks suppressed	Januaryƀƀ-Decemberƀ	
DD ZD DDD	2-digit day of month 1- or 2-digit day of mo. day of year (Julian day)	01-31 1-31 001-366	For output, leading zero is always suppressed. For input, ZD treated as DD.
HH ZH	2-digit hour 1- or 2-digit hour	00-23 0-23	For output, leading zero suppressed. For input, ZH treated as HH. If AP specified, valid values are 01-12.
MI	minute	00-59	
SS	second	00-59	
9 99 999			No rounding.
	tenths of a second	0-9	
	hundredths of a second	00-99	
	thousandths of a second	000-999	
AP ap A.P. a.p.	AM/PM indicator	AM or PM am or pm A.M. or P.M. a.m. or p.m.	AP affects HH/ZH field. For input, source string always folded to uppercase. For output, AP generates uppercase and ap generates lowercase.
W	1-char day-of-week	S, M, T, W, T, F, S	For input, Ws are ignored. For output, W generates uppercase and w generates lowercase. Output padded with blanks (unless Z specified) or truncated to match the number of Ws, up to 20.
WWW	3-char day, uppercase	SUN-SAT	
Www	3-char day, mixed case	Sun-Sat	
WWW...W	3-20 char day, uppercase	SUNDAY-SATURDAY	
Www...w	3-20 char day, mixed case	Sunday-Saturday	
WWWWWWWWWZ	trailing blanks suppressed	SUNDAYƀƀƀ-SATURDAYƀ	
Wwwwwwwwwz	trailing blanks suppressed	Sundayƀƀƀ-Saturdayƀ	

Table 33. Picture character terms used in picture strings for date and time services (continued)

Picture terms	Explanations	Valid values	Note
All others	Delimiters Constants	X'01'-X'FF' (X'00' reserved for Language Environment use)	For input, treated as delimiters between the month, day, year, hour, minute, second, and fraction of a second. For output, copied exactly as is to the target string. Constant designating year in Russia, Estonia, Latvia, Lithuania, and the Russian Federation. Constant designating time in Sweden.

Note: If a Z/z could be interpreted as belonging to the preceding character string *and* to the following string, then it is always considered part of the following string, even if it would be legal with the preceding string but illegal with the following string. For clarity, you should always use a delimiter to define which string the Z/z belongs with. See Table 34 for an example.

Table 34. Examples of picture strings recognized by date and time services

Picture strings	Examples	Notes®
YYMMDD YYYYMMDD YYYY-MM-DD <JJJJ> YY.MM.DD <CCCC> YY.MM.DD	880516 19880516 1988-05-16 *Showa* 63.05.16 *MinKow* 77.05.16	1988-5-16 would also be valid input. *Showa* is a Japanese Era name. *Showa* 63 equals 1988.
MMDDYY MM/DD/YY ZM/ZD/YY MM/DD/YYYY MM/DD/Y	050688 05 688 05/06/88 5/6/88 05/06/1988 05/06/8	Accepts imbedded blanks 1-digit year format (Y) valid for output only
DD.MM.YY DD-RRRR-YY DD MMM YY DD Mmmmmmmmmm YY ZD Mmmmmmmmmz YY Mmmmmmmmmz ZD, YYYY	09.06.88 09-VI -88 09 JUN 88 09 June 88 9 June 88 June 9, 1988	Z suppresses zeros/blanks
YY.DDD YYDDD YYYY/DDD	88.137 88137 1988/137	Julian date
YYMMDDHHMISS YYYYMMDDHHMISS YYYY-MM-DD HH:MI:SS.999 WWW, ZM/ZD/YY HH:MI AP Wwwwwwwwwz DD Mmm YYYY ZH:MI AP	880516204229 19880516204229 1988-05-16 20:42:29.046 MON, 5/16/88 08:42 PM Monday, 16 May 1988, 8:42 PM	Timestamp valid only for CEESECS and CEEDATM. If used with CEEDATE, time positions are left blank. If used with CEEDAYS, HH, MI, SS, and 999 fields are ignored.

Note: Lowercase characters must be used only for alphabetic picture terms.

Table 35. Japanese Eras used by date/time services when <JJJJ> is specified

First date of Japanese Era	Era name	Era name in IBM Japanese DBCS code	Valid year values
1868-09-08	Meiji	X'0E45A645840F'	01-45
1912-07-30	Taisho	X'0E455B45770F'	01-15
1926-12-25	Showa	X'0E45B3457A0F'	01-64
1989-01-08	Heisei	X'0E458D45BA0F'	01-999 (01 = 1989)

Appendix C. Controlling storage allocation

To generate a report of the storage a routine (or more specifically, an enclave) used during its run, specify the RPTSTG(ON) runtime option. The storage report, generated during enclave termination, provides statistics that can help you understand how space is being consumed as the enclave runs. If storage management tuning is desired, the statistics can help you set the corresponding storage-related runtime options for future runs.

Neither the storage report nor the corresponding runtime options include the storage that Language Environment acquires during early initialization, before runtime options processing, and before the start of space management monitoring.

Storage statistics

Attention: This section does not apply to AMODE 64 applications. For information about AMODE 64 statistics, see "Storage statistics for AMODE 64 applications."

The following runtime options control storage allocation:
- ANYHEAP
- BELOWHEAP
- HEAP
- HEAPPOOLS
- LIBSTACK
- STORAGE
- STACK
- THREADHEAP
- THREADSTACK

Ensure that these options are tuned appropriately to avoid performance problems. Tuning tips are provided in the *z/OS Language Environment Programming Guide*. For an example and complete description of the storage report, see *z/OS Language Environment Debugging Guide*.

Storage statistics for AMODE 64 applications

The following runtime options control storage allocation:
- HEAP64
- HEAPPOOLS
- HEAPPOOLS64
- IOHEAP64
- LIBHEAP64
- STACK64
- THREADSTACK64

Ensure that these options are tuned appropriately to avoid performance problems. Tuning tips are provided in the *z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode*. For an example and complete description of the storage report, see *z/OS Language Environment Debugging Guide*.

Appendix D. 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.

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 book documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Language Environment in z/OS.

Trademarks

Index

Product Number: 5650-ZOS

Printed in USA

SA38-0683-00