IBM Language Environment for VSE/ESA

Programming Reference

Version 1 Release 4 Modification Level 4

SC33-6685-05

IBM Language Environment for VSE/ESA

Programming Reference

Version 1 Release 4 Modification Level 4

SC33-6685-05

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

Sixth Edition (March 2005)

This edition applies to Version 1 Release 4 Modification Level 4 of IBM Language Environment for VSE/ESA, 5686-CF7, and to any subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the addresses given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

You may also send your comments by FAX or via the Internet:

Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456

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

Contents

Figures

Tables

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785, U.S.A.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

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

```
IBM Deutschland Informationssysteme GmbH
Department 0215
Pascalstr. 100
70569 Stuttgart
Germany
```

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

Programming Interface Information

This book is intended to help with application programming. This book documents General-Use Programming Interface and Associated Guidance Information provided by IBM Language Environment for VSE/ESA.

General-Use programming interfaces allow the customer to write programs that obtain the services of IBM Language Environment for VSE/ESA.

Trademarks and Service Marks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AT	Integrated Language Environment	VSE/ESA
C/370	Language Environment	z/OS
CICS	OS/390	zSeries
CICS/VSE	OS/400	z/VM
DB2	SAA	z/VSE
DFSORT	System/370	
IBM	VisualAge	

Microsoft, Windows, the Windows 95 logo, and Windows NT, are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

About This Book

z/VSE is the successor to IBM's VSE/ESA product. Many products and functions supported on z/VSE may continue to use VSE/ESA in their names.

z/VSE can execute in 31-bit mode only. It does not implement z/Architecture, and specifically does not implement 64-bit mode capabilities.

z/VSE is designed to exploit select features of IBM eServer zSeries hardware.

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

What Is LE/VSE?

LE/VSE is a set of common services and language-specific routines that provide a single run-time environment for applications written in *LE/VSE-conforming* versions of the C, COBOL, and PL/I high level languages (HLLs), and for many applications written in previous versions of COBOL. (For a list of LE/VSE-conforming languages, and a description of compatibility with previous versions of COBOL, see "LE/VSE-Conforming Languages" on page xvi.) LE/VSE also supports applications written in assembler language using LE/VSE-provided macros and assembled using High Level Assembler (HLASM).

Prior to LE/VSE, each programming language provided its own separate run-time environment. LE/VSE combines essential and commonly-used run-time services—such as message handling, condition handling, storage management, date and time services, and math functions—and makes them available through a set of interfaces that are consistent across programming languages. With LE/VSE, you can use one run-time environment for your applications, regardless of the application's programming language or system resource needs, because most system dependencies have been removed.

Services that work with only one language are available within language-specific portions of LE/VSE.

LE/VSE consists of:

- Basic routines for starting and stopping programs, allocating storage, communicating with programs written in different languages, and indicating and handling error conditions.
- Common library services, such as math services and date and time services, that are commonly needed by programs running on the system. These functions are supported through a library of callable services.
- Language-specific portions of the common run-time library.

LE/VSE is the implementation of Language Environment on the VSE platform. Language Environment is offered on other platforms: on z/OS and VM, and on OS/400 as Integrated Language Environment.

LE/VSE-Conforming Languages

An LE/VSE-conforming language is any HLL that adheres to the LE/VSE common interface. Table 1 lists the LE/VSE-conforming language compiler products you can use to generate applications that run with LE/VSE Release 4.

Table 1. LE/VSE-Conforming Languages

Language	LE/VSE-Conforming Language	Minimum Release
C	IBM C for VSE/ESA	Release 1
COBOL	IBM COBOL for VSE/ESA	Release 1
PL/I	IBM PL/I for VSE/ESA	Release 1

Any HLL not listed in Table 1 is known as a *non-LE/VSE-conforming* or, alternatively, a *pre-LE/VSE-conforming* language. Some examples of non-LE/VSE-conforming languages are: C/370, DOS/VS COBOL, VS COBOL II, and DOS PL/I.

Only the following products can generate applications that run with LE/VSE:

- LE/VSE-conforming languages
- HLASM using LE/VSE-provided macros (for details, see *LE/VSE Programming Guide*)
- DOS/VS COBOL and VS COBOL II, with some restrictions (see LE/VSE Compatibility with Previous Versions of COBOL below)

LE/VSE Compatibility with Previous Versions of COBOL

Although DOS/VS COBOL and VS COBOL II are non-LE/VSE-conforming languages, many applications generated with these compilers can run with LE/VSE without recompiling or relink-editing. For details about compatibility, see *LE/VSE Run-Time Migration Guide*.

VS COBOL II can also dynamically call some LE/VSE date and time callable services. For details, see *LE/VSE Programming Reference*.

Terms Used in This Book

Unless otherwise stated, the following terms are used in this book to refer to the specified languages:

Term...	Refers to the language supported by...
C	The IBM C for VSE/ESA compiler
COBOL	The IBM COBOL for VSE/ESA and VS COBOL II compilers
PL/I	The IBM PL/I for VSE/ESA compilers

For a list of LE/VSE-conforming language compilers, see "LE/VSE-Conforming Languages."

How to Read the Syntax Diagrams

The following rules apply to the notation used in the syntax diagrams contained in this book:

- Read the syntax diagrams from left to right, top to bottom following the path of the line.
- Each syntax diagram begins with a double arrowhead (►►).
- An arrow (→►) at the end of a line indicates that the option, service, or macro syntax continues on the next line. A continuation line begins with an arrow (►→).
- If a syntax diagram contains too many items or groups to fit in the diagram, the syntax is shown by a main syntax diagram and one or more syntax fragments. A syntax fragment is referred to in the main diagram by its fragment name between two vertical bars (|).

Each syntax fragment appears below the main syntax diagram, and begins and ends with a vertical bar (|). A heading above the fragment indicates the name of the fragment.

Read each syntax fragment as though it were imbedded in the main syntax diagram.

- IBM-supplied default keywords appear **above** the main path or options path (see the sample on page xvii). In the parameter list, IBM-supplied default choices are underlined.
- Keywords appear in nonitalic capital letters and should be entered exactly as shown. However, some keywords may be abbreviated by truncation from the right as long as the result is unambiguous. In this case, the unambiguous truncation is shown in capital letters in the keyword, for example:

  ```
  ANyheap
  ```

- Words in lowercase letters represent user-defined parameters or suboptions.
- Enter parentheses, arithmetic symbols, colons, semicolons, commas, and greater-than signs where shown.
- Required parameters appear on the same horizontal line (the main path) as the option, service, or macro:

- If you can choose from two or more parameters, the choices are stacked one above the other.

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

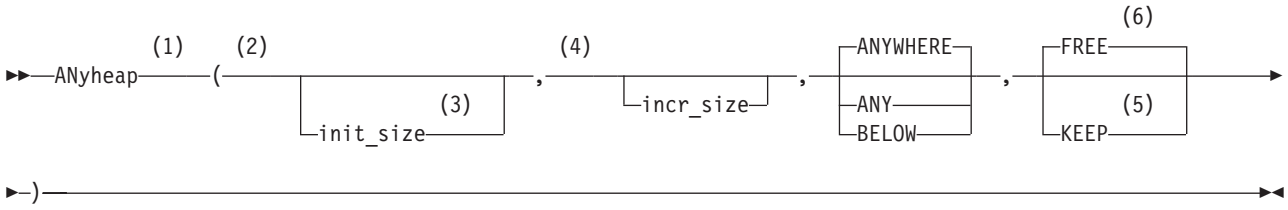
  If you *must* choose one of the items, one item of the stack appears on the main path:

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

  OR

- A comma or semicolon included in the repeat symbol indicates a separator that you must include between repeated parameters. These separators must be coded where shown.
- When entering commands, parameters and keywords must be separated by at least one blank if there is no intervening punctuation.
- A double arrow (→◄) at the end of a line indicates the end of the syntax diagram.

The following example demonstrates how to read the syntax notation. Numbers in the example correspond to explanations supplied below the example.



Notes:

1 Keyword with minimum unambiguous truncation shown in capital letters

2 Opening parenthesis (must be specified if any parameters are specified)

3 Optional parameter

4 Comma (must be specified if there are parameters that follow)

5 Optional keyword

6 Optional keyword (IBM-supplied default)

Where to Find More Information

These are the manuals that describe LE/VSE:

Table 2. LE/VSE Publications

Publication	Form Number
LE/VSE Fact Sheet	GC33-6679
LE/VSE Concepts Guide	GC33-6680
LE/VSE Customization Guide	SC33-6682
LE/VSE Programming Guide	SC33-6684
LE/VSE Programming Reference	SC33-6685
LE/VSE C Run-Time Programming Guide	SC33-6688
LE/VSE C Run-Time Library Reference	SC33-6689
LE/VSE Debugging Guide and Run-Time Messages	SC33-6681
LE/VSE Writing Interlanguage Communication Applications	SC33-6686
LE/VSE Run-Time Migration Guide	SC33-6687
LE/VSE Licensed Program Specifications	GC33-6683

These are the z/VSE manuals to which you might need to refer:

Table 3. z/VSE Publications

Publication	Form Number
z/VSE Administration	SC33-8224
z/VSE Messages and Codes, Volume 1	SC33-8226
z/VSE Messages and Codes, Volume 2	SC33-8227
z/VSE Messages and Codes, Volume 3	SC33-8228
z/VSE Planning	SC33-8221
z/VSE System Control Statements	SC33-8225
z/VSE System Macros Reference	SC33-8230
z/VSE System Macros User's Guide	SC33-8236
z/VSE System Upgrade and Service	SC33-8223
VSE/VSAM User's Guide and Application Programming	SC33-8246
VSE/VSAM Commands	SC33-8245
TCP/IP for VSE/ESA IBM Program Setup and Supplementary Information	SC33-6601

These are the manuals that describe IBM C for VSE/ESA:

Table 4. IBM C for VSE/ESA Publications

Publication	Form Number
Licensed Program Specifications	GC09-2421
Installation and Customization Guide	GC09-2422
Migration Guide	SC09-2423

Table 4. IBM C for VSE/ESA Publications (continued)

Publication	Form Number
User's Guide	SC09-2424
Language Reference	SC09-2425
Diagnosis Guide	GC09-2426

These are the manuals that describe IBM COBOL for VSE/ESA:

Table 5. IBM COBOL for VSE/ESA Publications

Publication	Form Number
General Information	GC33-6679
Licensed Program Specifications	GC33-6680
Migration Guide	SC33-6682
Installation and Customization Guide	GC33-6680
Programming Guide	SC33-6684
Language Reference	SC33-6685
Diagnosis Guide	SC33-6684
Reference Summary	SX26-3834

These are the manuals that describe IBM PL/I for VSE/ESA:

Table 6. IBM PL/I for VSE/ESA Publications

Publication	Form Number
Fact Sheet	GC26-8052
Programming Guide	SC26-8053
Language Reference	SC26-8054
Licensed Program Specifications	GC26-8055
Migration Guide	SC33-6684
Installation and Customization Guide	SC26-8057
Diagnosis Guide	SC26-8058
Compile-Time Messages and Codes	SC26-8059
Reference Summary	SX26-3836

These are the manuals that describe Debug Tool for VSE/ESA:

Table 7. Debug Tool for VSE/ESA Publications

Publication	Form Number
User's Guide and Reference	SC26-8797
Installation and Customization Guide	SC26-8798
Fact Sheet	GC26-8925

You might also refer to the ...

z/VSE Home Page

z/VSE has a home page on the World Wide Web, which offers up-to-date information about VSE-related products and services, new z/VSE functions, and other items of interest to VSE users.

You can find the z/VSE home page at:

`http://www.ibm.com/servers/eserver/zseries/zvse/`

Softcopy Publications

The following collection kit contains the LE/VSE and LE/VSE-conforming language product publications:

VSE Collection, SK2T-0060

Summary Of Changes

This section describes the changes introduced with the current and previous two editions of this manual.

Changes Introduced with Sixth Edition (March 2005)

These are the most important changes introduced with the sixth edition of this manual (covering LE/VSE 1.4.4):

- The name **VSE/ESA** has now changed to **z/VSE**. However, the names of many features and programs related to z/VSE remain unchanged (such as IBM Language Environment for VSE/ESA, IBM COBOL for VSE/ESA, Debug Tool for VSE/ESA, or CICS Transaction Server for VSE/ESA).
- The sample run-time options report that shows the use of the RPTOPTS(ON) run-time option has enhanced. For details, see Figure 2 on page 38.
- The sample storage report that shows the use of the RPTSTG(ON) run-time option has enhanced. For details, see Figure 3 on page 39.
- The LSTQ sub-option of the TERMTHDACT run-time option has enhanced, and is also available in batch. For details, see 45.
- The sample run-time options report now shows the use of the LSTQ sub-option. For details, see 45.
- A description of the LE/VSE euro support has been included. For details, see "CEE5MCS and Euro Support" on page 83.
- Callable service CEE5PRM has been changed back to its previous functionality, so that it can be used with a buffer of length 80 characters. For details, see "CEE5PRM—Query Parameter String" on page 87.
- Callable service CEE5PRML is new. You can use it with a buffer of length 300 characters. For details, see "CEE5PRML—Query Parameter String (Long Parameters)" on page 88.
- Callable service CEE5TSTG is new. You can use it to test the access that is available to a specified storage address. For details, see "CEE5TSTG—Test Access Authority for a Supplied Storage Address" on page 95.
- Some currency symbols have changed due to the introduction of the euro currency. These new codes are listed in Appendix A, "IBM-Supplied Country/Region Code Defaults," on page 227.

Changes Introduced with Fifth Edition (March 2003)

The fifth edition of the *LE/VSE Programming Reference* (March 2003), contained these changes:

- You can now use the CEELOPT macro to modify the default CLASS and DISP for output sent to the VSE/POWER LSTQ. For details, see the description of the LSTQ sub-option on page 45.
- Important information for the application programmer was included concerning the use of the *clean-up* parameter of the CEE5ABD callable service. For details, see "Usage Notes" on page 62.
- Support for JCL invocation strings that are up to 300 characters in length. For details, see "CEE5PRM—Query Parameter String" on page 87.

Changes Introduced with Fourth Edition (March 2002)

The fourth edition of the *LE/VSE Programming Reference*, SC33-6685-03 (March 2002), contained these changes:

- Chapter 2, "LE/VSE Run-Time Options," on page 19 was updated with:
  - Run-time options HEAPCHK and RETZERO, that were new with LE/VSE 1.4.0 (June 1999).
  - Other changes to run-time options that were included in LE/VSE 1.4.0 (June 1999) and LE/VSE 1.4.1 (September 2000).

- Chapter 3, "LE/VSE Callable Services," on page 57 was updated with:
  - The CEE5DLY Enclave Delay callable service that was new with LE/VSE 1.4.2 (December 2001).
  - These callable services that were new with LE/VSE 1.4.1 (September 2000):
    - CEE5GRO— Get Offset of Most Recent Condition
    - CEE5SRP— Set Resume Point
    - CEEMRCE— Move Resume Cursor Explicit
    - CEESICLR— Bit Clear (Manipulation Routine)
    - CEESISET— Bit Set (Manipulation Routine)
    - CEESISHF— Bit Shift (Manipulation Routine)
    - CEESITST— Bit Test (Manipulation Routine)

Chapter 1. LE/VSE Quick Reference Tables

The following tables are a quick reference of the LE/VSE run-time options, callable services, and math services. They list the syntax of the options and services and briefly state their function.

For detailed descriptions of these options and services, refer to their respective chapters in this book.

Run-Time Options

Table 8. Run-Time Options Quick Reference

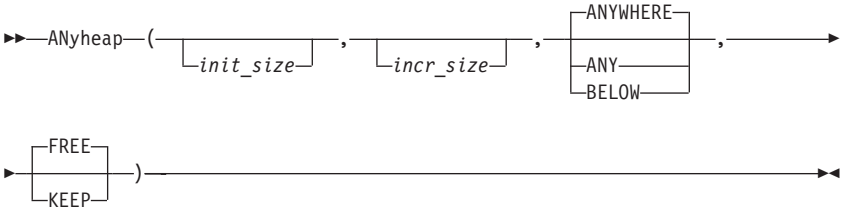
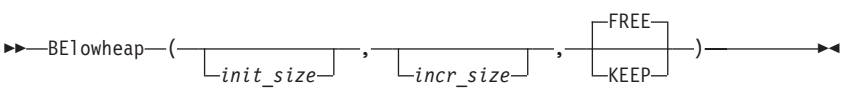
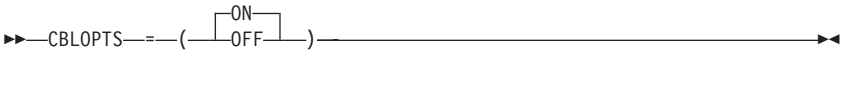
Run-Time Options	Function	Page
	Exempts a specified VSE cancel code, program-interruption code, or user abend code from LE/VSE condition handling.	19
	Sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater.	20
	(COBOL only) Invokes the access method services (AMS) for VSAM key-sequenced data sets (KSDS) and relative-record data sets (RRDS) to complete the file and index definition procedures for COBOL routines.	21
	Indicates whether an application does or does not run entirely in AMODE(31).	22
	Controls allocation of library heap storage not restricted to below the 16MB line.	23
	(C only) Specifies whether arguments on the command line are to be parsed in the usual C format.	24
	Controls allocation of library heap storage below the 16MB line.	24
	(COBOL only) Specifies the format of the argument string on application invocation when the main program is COBOL.	25

Table 8. Run-Time Options Quick Reference (continued)

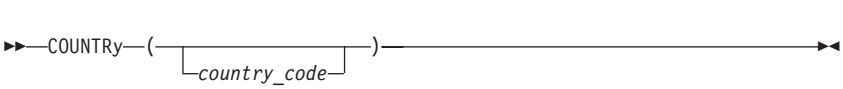
Run-Time Options	Function	Page
	(COBOL only) Controls whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL subprogram is called.	25
	(COBOL only) Indicates whether "checking errors" within an application should be detected.	26
	Specifies the default formats for date, time, currency symbol, decimal separator, and the thousands separator based on a country.	26
	(COBOL only) Activates the COBOL batch debugging features specified by the "debugging lines" or the USE FOR DEBUGGING declarative.	27
	Limits the extent to which conditions can be nested.	27
	(C only) Specifies the operating environment that your C application runs under.	28
	(C only) Sets the initial values for the environment variables specified in *string*.	28
	Specifies how many conditions of severity 2, 3, and 4 can occur per thread before an enclave terminates abnormally.	29
	(C only) Specifies whether run-time options can be specified on the command line.	30

Table 8. Run-Time Options Quick Reference (continued)

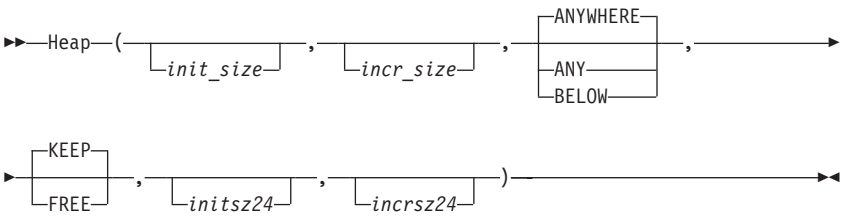
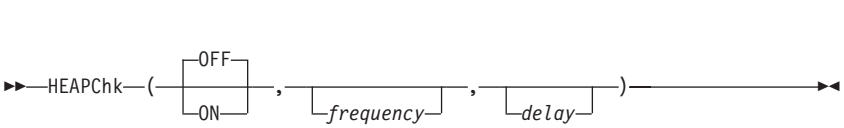
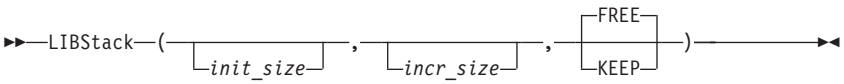
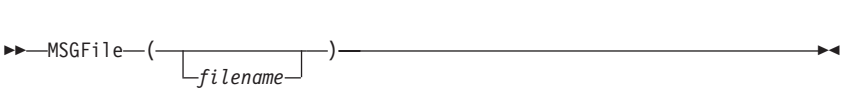
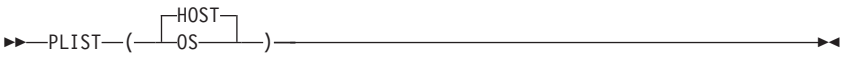
Run-Time Options	Function	Page
	Controls allocation of the heaps.	30
	Provides a checking facility to verify that the heap storage has not been damaged.	32
	Controls the allocation of the thread's library stack storage.	32
	Specifies the *filename* of the run-time diagnostics file.	33
	Specifies the number of ISI blocks allocated on a per-thread basis during execution.	34
	Specifies the national language to use for the run-time environment.	35
	(C only) Specifies the format of the invocation arguments received by your C application when it is invoked.	36
	(C only) Specifies whether redirections for `stdin`, `stderr`, and `stdout` are allowed from the command line.	36
	(COBOL only) Ensures that, if the run unit does not abend or terminate abnormally, the user return code will be set to zero regardless of the contents of register 15 or the RETURN-CODE special register.	36

Table 8. Run-Time Options Quick Reference (continued)

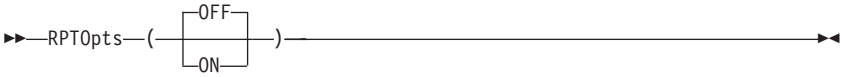
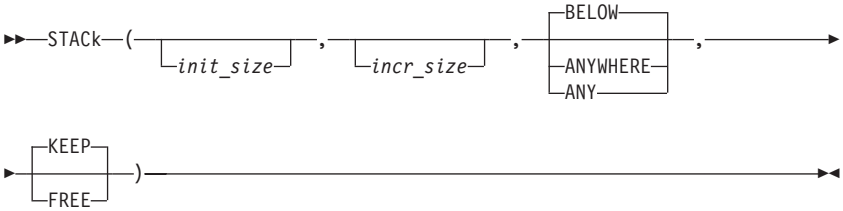
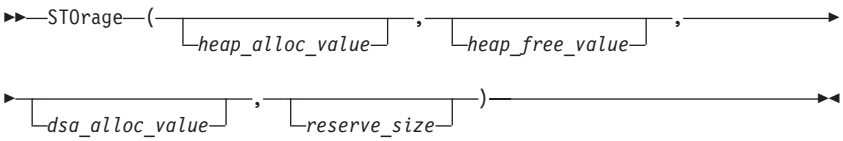
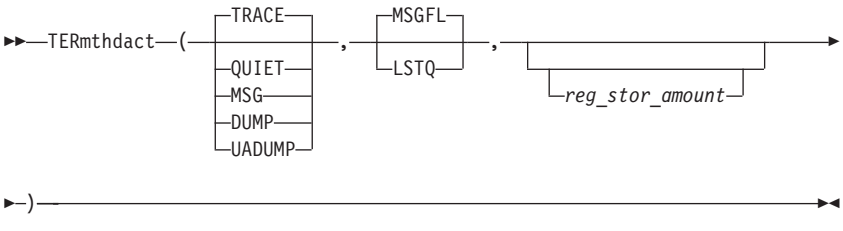
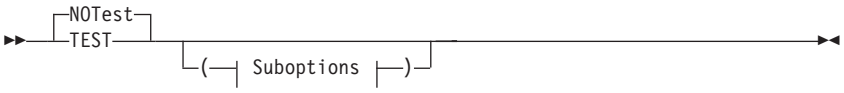
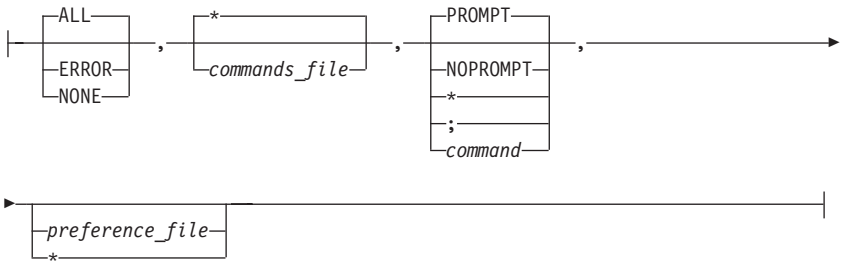
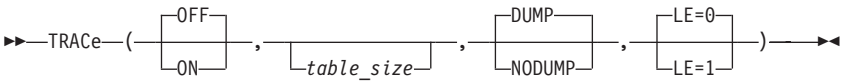
Run-Time Options	Function	Page
	Specifies that a report of the run-time options in use by the application be generated.	37
	Specifies that a report of the storage used by the application be generated at the end of execution.	38
	Initializes the run-time environment to be reusable when the first COBOL program is invoked.	40
	Controls the allocation and management of thread-level stack storage.	40
	Controls the value of storage that is allocated and freed.	42
	Sets (1) the level of information produced due to an unhandled error of severity 2 or greater, (2) the information output destination, and (3) the amount of storage to be dump around each general purpose register.	44

Table 8. Run-Time Options Quick Reference (continued)

Run-Time Options	Function	Page
	Specifies that a debug tool is to be given control according to the suboptions specified.	46
Suboptions:		
		
	Determines whether LE/VSE run-time library tracing is active.	48
	Specifies if LE/VSE is to handle program exceptions and abends, and how much LE/VSE is to be involved.	48
	(COBOL only) Sets the eight UPSI switches on or off. Affects only COBOL programs.	50
	Registers a user condition handler at stack frame 0.	51
	Specifies whether an exponent underflow causes a program interrupt.	52

Condition Handling Callable Services

Table 9. Condition Handling Callable Services

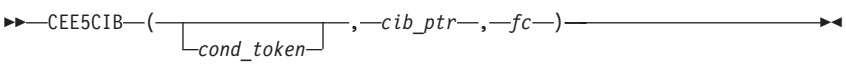
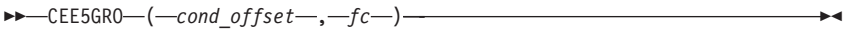
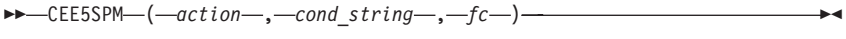
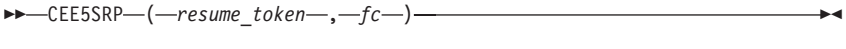
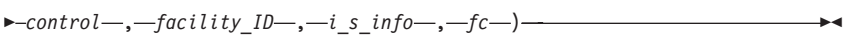
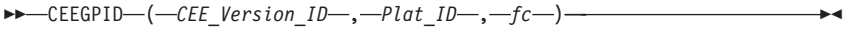
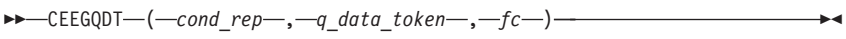
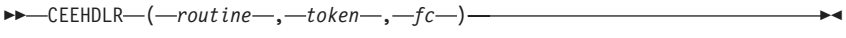
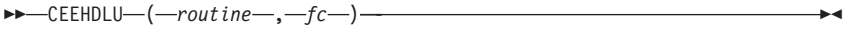
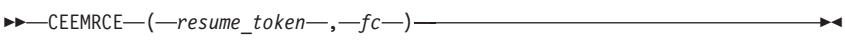
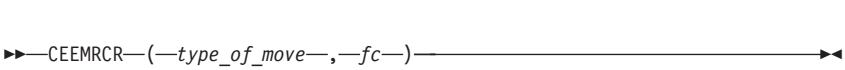
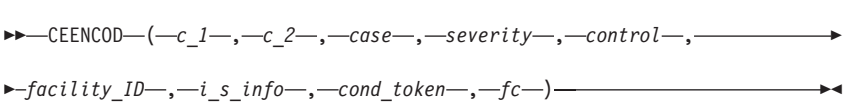
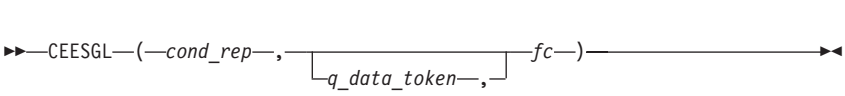
Callable Service	Function	Page
	Returns a pointer to a condition information block (CIB) associated with a given condition token.	63
	Gets the name of the most current LE/VSE-conforming routine in which a condition occurred.	76
	Returns the offset of the most recent condition, within a failing routine.	78
	Queries and modifies the enablement of LE/VSE hardware conditions.	90
	Sets the resume point at the next instruction in the calling routine.	94
	Decomposes or alters an existing condition token.	117
	Retrieves the version ID and the platform ID of the version and platform of LE/VSE currently in use.	138
	Provides a mechanism by which application code (in particular, PL/I ON-units) can retrieve the *q_data_token* from the Instance Specific Information (ISI).	140
	Registers a user condition handler for the current stack frame.	146
	Unregisters a user-written condition handler for the current stack frame.	149
	Computes the initial condition token for the current condition information block.	153

Table 9. Condition Handling Callable Services (continued)

Callable Service	Function	Page
	Resumes execution of a user routine at the location established by CEE5SRP.	164
	Moves the resume cursor to a position relative to the current position of the handle cursor.	166
	Dynamically constructs a condition token.	173
	Raises, or signals, a condition to the condition manager.	193

Date and Time Callable Services

Table 10. Date and Time Callable Services

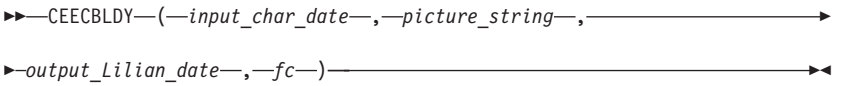
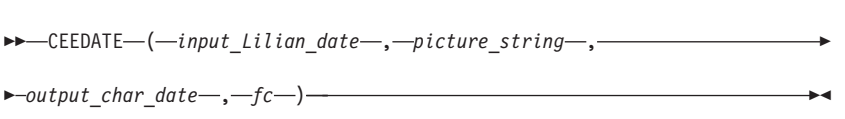
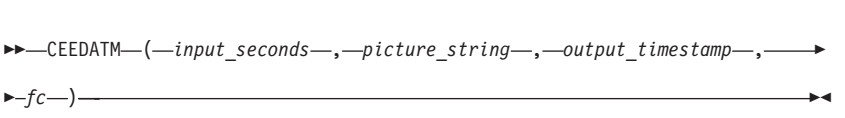
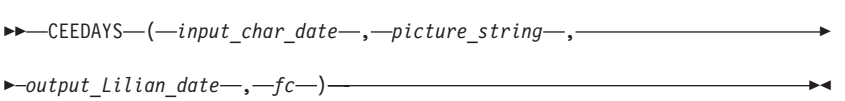
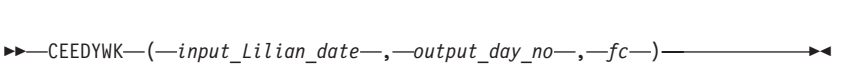
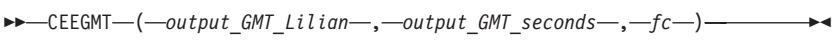
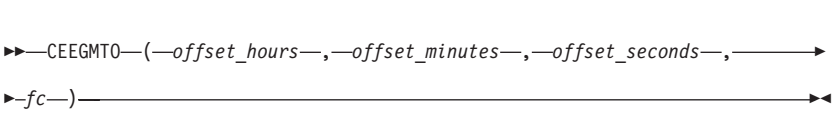
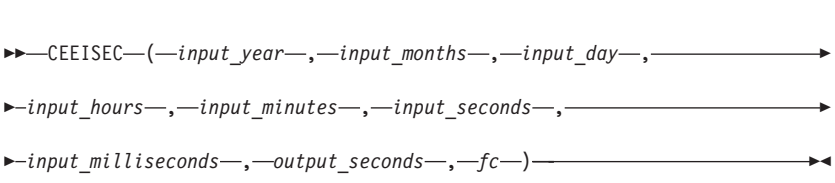
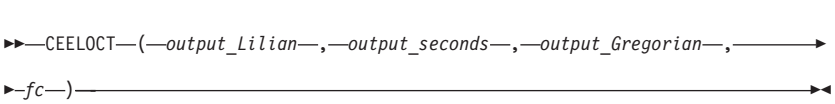
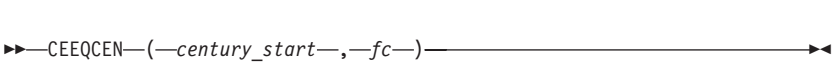
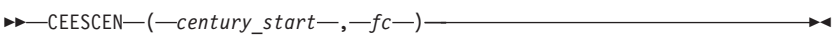
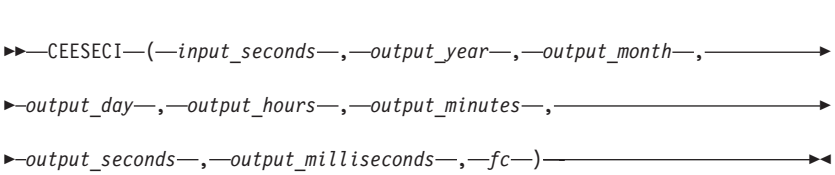
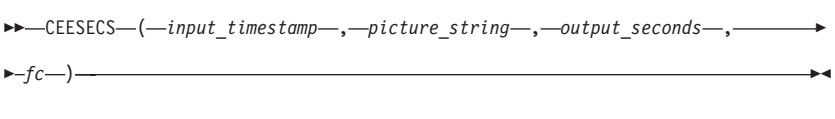
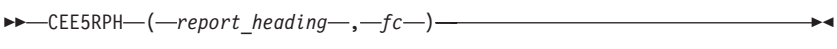
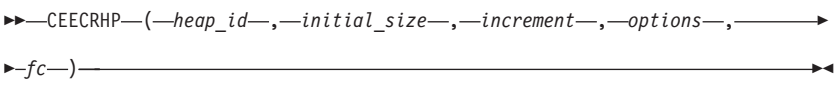
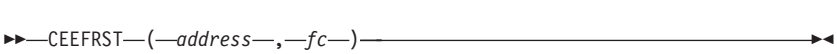
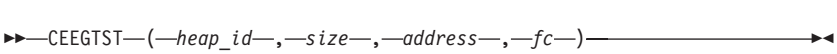
Callable Service	Function	Page
	Converts a string representing a date to a COBOL integer format.	98
	Converts a number representing a Lilian date to a date written in character format.	108
	Converts a number representing the number of seconds since 00:00:00 14 October 1582 to character format.	110
	Converts a string representing a date to a Lilian format.	113
	Calculates the day of the week on which a Lilian date falls.	121

Table 10. Date and Time Callable Services (continued)

Callable Service	Function	Page
	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.	135
	Computes values to the calling routine that represent the difference between the local system time and Greenwich Mean Time.	137
	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.	151
	Gets the current local time in three formats.	158
	Queries the century within which LE/VSE assumes two-digit year values lie.	176
	Sets the century in which LE/VSE assumes two-digit year values lie.	182
	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.	185
	Converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582.	188

Dynamic Storage Callable Services

Table 11. Dynamic Storage Callable Services

Callable Service	Function	Page
	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) run-time options.	89
	Defines additional heaps.	103
	Changes the size of a previously allocated storage element, while preserving its contents.	105
	Discards an entire heap that you created previously with a call to CEECRHP.	119
	Frees storage previously allocated by CEEGTST or a language intrinsic function.	129
	Allocates storage from a heap whose ID you specify.	143

General Callable Services

Table 12. General Callable Services

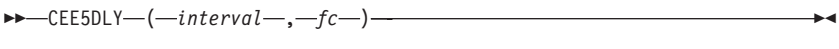
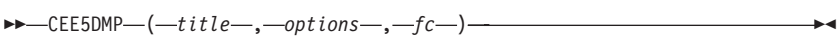
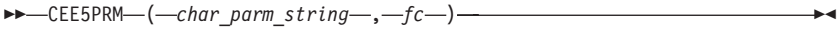
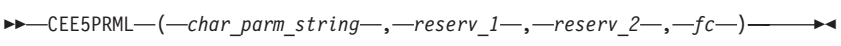
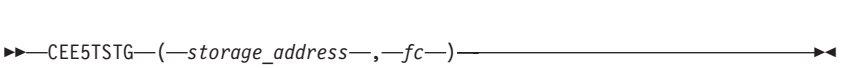
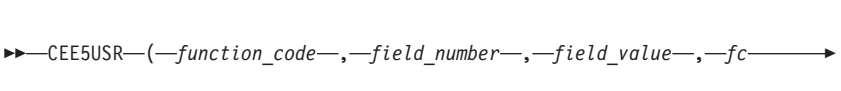
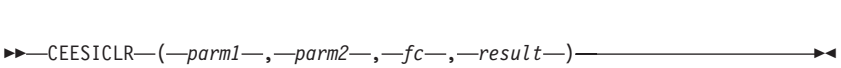
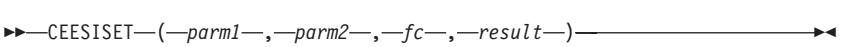
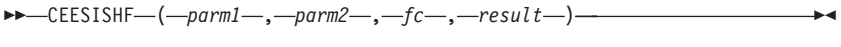
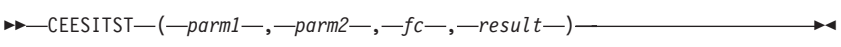
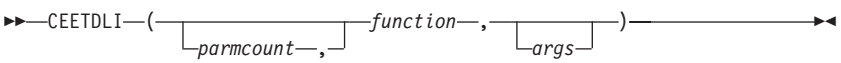
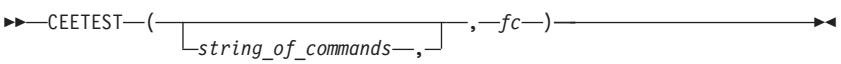
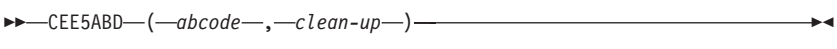
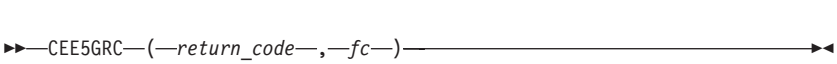
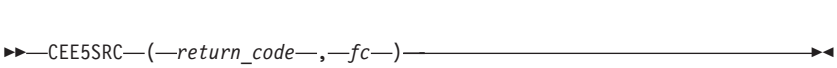
Callable Service	Function	Page
	Suspends processing of the active enclave for the specified number of seconds.	67
	Generates a dump of the run-time environment of LE/VSE and the member language libraries.	69
	Returns to the calling routine an 80-byte argument string that was specified at invocation of the program.	87

Table 12. General Callable Services (continued)

Callable Service	Function	Page
	Returns to the calling routine a 300-byte argument string that was specified at invocation of the program.	88
	Tests for the access that is available to a specified storage address. This access can be no-access permitted, read-only access, or update access.	95
	Sets or queries one of two 4-byte fields known as the user area fields.	96
	Generates a sequence of uniform pseudo-random numbers between 0.0 and 1.0 using the multiplicative congruential method with a user-specified seed.	181
	Returns a copy of its *parm1* input, but with a single bit selectively set to 0.	196
	Returns a copy of its *parm1* input, but with a single bit selectively set to 1.	196
	Returns a copy of its *parm1* input, right- or left-shifted by the number of bits indicated by *parm2*.	196
	Selectively tests a bit in its *parm1* input to determine if the bit is on.	197
	Provides an interface to DL/I.	199
	Invokes a debug tool, such as Debug Tool for VSE/ESA.	200

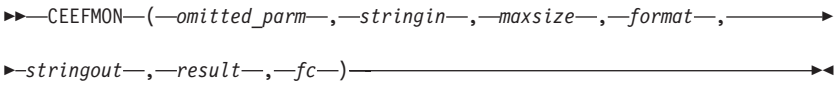
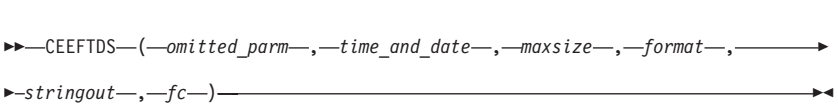
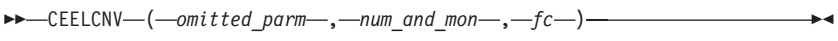
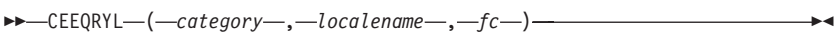
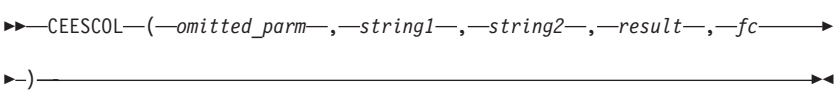
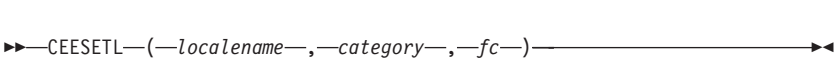
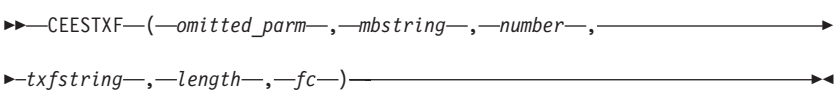
Initialization and Termination Services

Table 13. Initialization and Termination Services

Callable Service	Function	Page
	Terminates the enclave with an abend. The abend can be issued either with or without clean-up.	62
	Gets the enclave return code.	73
	Sets the enclave return code.	94

Locale Callable Services

Table 14. Locale Callable Services

Callable Service	Function	Page
	Formats monetary strings.	125
	Formats time and date into a character string.	132
	Queries locale numeric conventions.	155
	Queries locale date and time conventions and returns the specified format information.	178
	Queries the active locale environment.	180
	Compares the collation weights of two strings.	184
	Sets the locale operating environment.	191
	Transforms string characters into collation weights.	197

Math Services

Table 15. Math Services

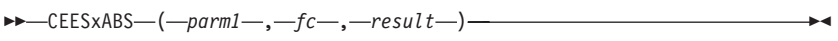
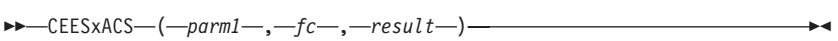
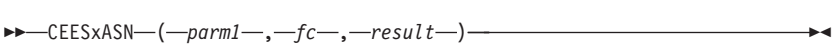
Math Service	Function	Page
	Absolute value	204
	Arccosine	205
	Arcsine	205

Table 15. Math Services (continued)

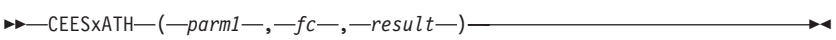
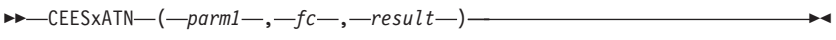
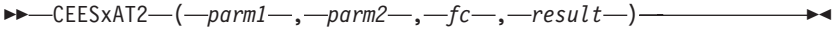
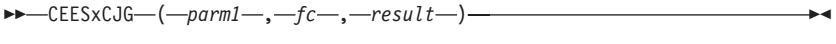
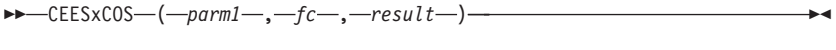
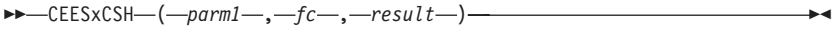
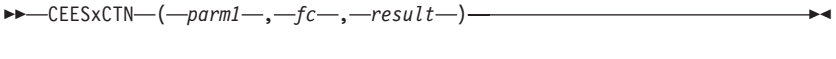
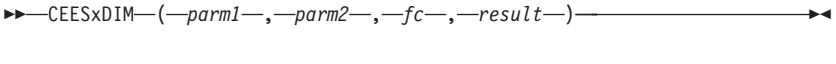
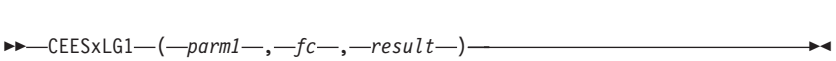
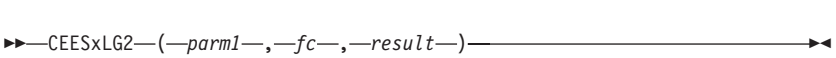
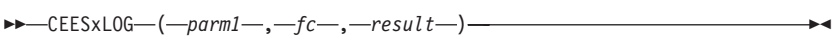
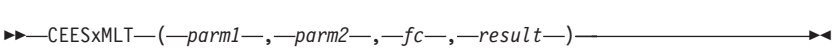
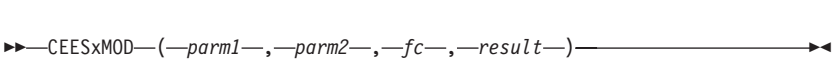
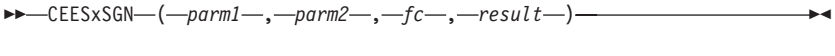
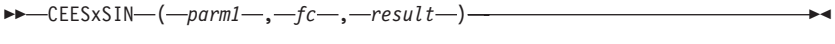
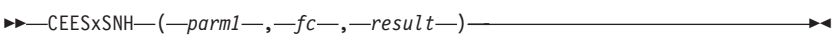
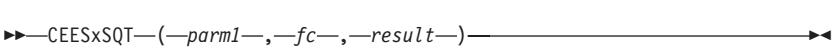
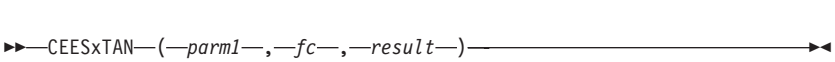
Math Service	Function	Page
	Hyperbolic arctangent	206
	Arctangent	206
	Arctangent of two arguments	207
	Conjugate complex	207
	Cosine	208
	Hyperbolic cosine	209
	Cotangent	209
	Positive difference	210
	Division	210
	Error function complement	211
	Error function	211
	Exponential (base e)	212
	Gamma function	213
	Imaginary part of complex	213
	Truncation	214
	Log gamma function	214
	Logarithm base 10	215
	Logarithm base 2	215

Table 15. Math Services (continued)

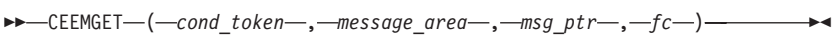
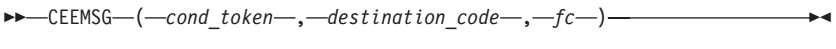
Math Service	Function	Page
	Logarithm base e	216
	Floating-point complex multiplication	216
	Modular arithmetic	216
	Nearest integer	217
	Nearest whole number	218
	Transfer of sign	218
	Sine	219
	Hyperbolic sine	219
	Square root	220
	Tangent	221
	Hyperbolic tangent	222
	Exponential (**)	222

Message Handling Callable Services

Table 16. Message Handling Callable Services

Callable Service	Function	Page
	Stores message insert data.	101

Table 16. Message Handling Callable Services (continued)

Callable Service	Function	Page
	Gets, formats, and stores in a buffer a message corresponding to a condition token returned from a callable service or passed to a user-written condition handler.	160
	Dispatches a user-defined message to the message string file.	162
	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.	171

National Language Support Callable Services

Table 17. National Language Support and National Language Architecture Callable Services

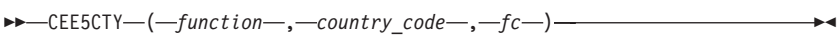
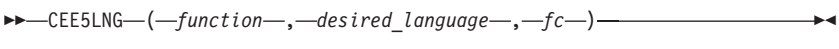
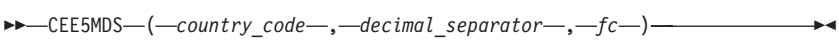
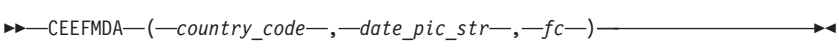
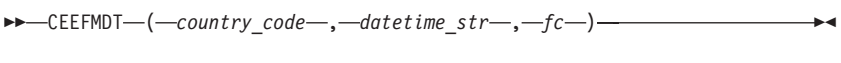
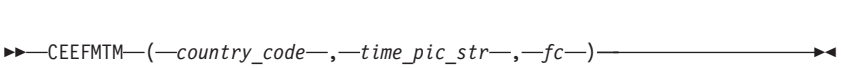
Callable Service	Function	Page
	Changes or queries the current national country setting.	65
	Changes or queries the current national language.	79
	Gets the default currency symbol for a country specified in *country_code*.	82
	Gets the default decimal separator for the country specified in *country_code*.	84
	Gets the default thousands separator for the country that you specify in *country_code*	85
	Gets the default date picture string for a country specified in *country_code*.	123

Table 17. National Language Support and National Language Architecture Callable Services (continued)

Callable Service	Function	Page
	Gets the default date and time picture strings for the country specified in *country_code*.	124
	Gets the default time picture string for the country specified in *country_code*.	128

Chapter 2. LE/VSE Run-Time Options

This chapter describes the LE/VSE run-time options, their syntax, and their usage.

The syntax diagrams show how the run-time options are normally used in the PARM JCL statement. However, for run-time options that cannot be changed using the PARM JCL statement, the alternative syntax is shown.

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 LE/VSE option is also indicated in its syntax diagram with capital letters (for example, ABPerc indicates that ABP is the minimum abbreviation).

Note: You can use these abbreviations as an override in the PARM JCL.

For a list of LE/VSE run-time options, see Chapter 1, "LE/VSE Quick Reference Tables," on page 1.

How to Specify Run-Time Options

When specifying run-time options, use commas to separate any suboptions of those options. If you do not specify a suboption, you must still specify the comma to indicate its omission, for example HEAPCHK(,1,0). Trailing commas, however, are not required; HEAPCHK(OFF) is valid. If you do not specify any suboptions, either of the following is valid: HEAPCHK or HEAPCHK().

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

LE/VSE Programming Guide describes the various ways in which you can specify LE/VSE run-time options and program arguments. Refer to it for the syntax of run-time options and user arguments in the invocation command string.

ABPERC

ABPERC exempts a specified VSE cancel code, program-interruption code, or user abend code from LE/VSE condition handling, and causes an operating system request to be issued to terminate the enclave.

The ABPERC option is a debugging aid that can be used by an application that runs with TRAP set to ON. This provides LE/VSE semantics for everything except one VSE cancel, program interruption, or user abend, whose code you specify.

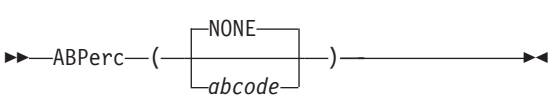
When you run with ABPERC and encounter the specified VSE cancel, interruption, or user abend:

- User condition handlers are not enabled.
- No storage report or run-time options report is generated.
- No LE/VSE messages or LE/VSE dump output is generated.
- The assembler user exit is not driven for enclave termination.
- The abnormal termination exit (if there is one) is not driven.
- Files opened by HLLs are not closed by LE/VSE, so records might be lost.
- Resources acquired by LE/VSE are not freed.
- The debug tool is not notified of the error.

You can also specify a list of VSE cancel codes, interruption codes, and user abend codes in the CEEBXITA assembler user exit for the condition manager to exempt from LE/VSE condition handling.

IBM-Supplied Default: ABPERC(NONE)

Syntax



NONE
Specifies that all abnormal terminations are handled according to LE/VSE condition handling semantics.

abcode
Specifies the VSE cancel code, program-interruption code, or user abend code to be exempted from LE/VSE condition handling.

abcode can be specified as:

S*hh* A VSE cancel code where ***hh*** is the hexadecimal cancel code.

I*hh* A VSE interruption code where ***hh*** is the hexadecimal interruption code.

U*dddd* A user abend code where ***dddd*** is a decimal user-issued abend code.

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

You can identify only one VSE cancel code, program-interruption code, or abend code with this option.

Usage Notes

- LE/VSE ignores ABPERC(S20). The VSE cancel code 20 indicates a program check has occurred. In this instance, LE/VSE condition handling semantics are in effect. You can, however, specify one program check interruption code, in the form I*hh*, to be exempted from LE/VSE condition handling.
- CICS consideration—ABPERC is ignored under CICS.

For More Information

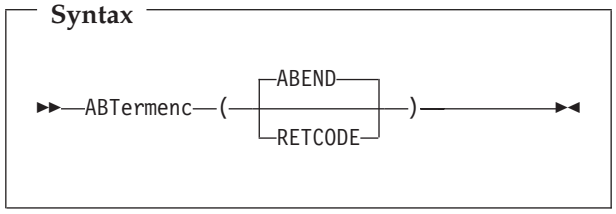
- For more information about the CEEBXITA assembler user exit, see *LE/VSE Programming Guide*.
- For more information about VSE cancel codes, see *z/VSE Messages and Codes*
- For a list of program-interruption codes, see the *Principles of Operations* manual for your machine.

ABTERMENC

ABTERMENC sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater. TRAP(ON) must be in effect for ABTERMENC to have an effect when the unhandled condition is a program check or an abend. ABTERMENC is always in effect for unhandled conditions raised by the CEESGL callable service, regardless of the setting of the TRAP option.

IBM-Supplied Default: ABTERMENC(ABEND)

Syntax



ABEND
Specifies that LE/VSE terminates the enclave with an abend, regardless of the setting of the CEEAUE_ABND flag by the assembler user exit. In the batch environment, LE/VSE produces run-time message CEE3321C or CEE3322C, and issues an operating system request to terminate the enclave. In the CICS environment, LE/VSE issues an EXEC CICS ABEND. The setting of the CEEAUE_ABND flag affects the abend processing, as follows:

When CEEAUE_ABND is off, the following occurs:

- Abend code: LE/VSE sets an abend code value that depends on the type of unhandled condition.
- Reason code: LE/VSE sets a reason code value that depends on the type of unhandled condition.
- Abend dump attribute: LE/VSE does not request a system dump.

When CEEAUE_ABND is on, LE/VSE uses values set by the assembler user exit to determine abend processing:

- Abend code: Value of the CEEAUE_RETURN parameter of the assembler user exit.
- Reason code: Value of the CEEAUE_REASON parameter of the assembler user exit.
- Abend dump attribute: LE/VSE requests a system dump only if the assembler user exit sets CEEAUE_DUMP to ON.

RETCODE
Specifies that the enclave terminates with a normal return code and reason code.

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

- If the assembler user exit does not set the CEEAUE_ABND flag to ON during enclave termination, LE/VSE returns to its caller with a return code and a reason code.
- If the assembler user exit sets the CEEAUE_ABND flag to ON during enclave termination, LE/VSE terminates the enclave with an abend. In the batch environment, LE/VSE produces the run-time message CEE3322C, and issues an operating system request to terminate the enclave. In the CICS environment, LE/VSE issues an EXEC CICS ABEND.

  LE/VSE sets the abend and reason code for the abend to equal the values of assembler-user-exit parameters, as follows:

  – Abend code: Value of the CEEAUE_RETURN parameter of the assembler user exit. If the assembler user exit does not modify the CEEAUE_RETURN value, LE/VSE sets an abend code that maps to the severity of the condition and to the user return code.
  – Reason code: Value of the CEEAUE_REASON parameter of the assembler user exit.

Usage Notes

- COBOL consideration—For compatibility with pre-LE/VSE-conforming COBOL, ABEND is the recommended setting for COBOL customers.
- PL/I consideration—For compatibility with DOS PL/I, ABEND is the recommended setting for PL/I customers.
- DB2 considerations—ABEND is the recommended setting for DB2 users. This ensures that error conditions are mirrored back to DB2 to enable ROLLBACK. See also the chapter "Running Applications with DB2" of the *LE/VSE Programming Guide*.
- DL/I considerations—ABEND is the recommended setting for DL/I users.

For More Information

- For information about return code calculation, CEEAUE_RETURN, CEEAUE_ABND, and CEEBXITA assembler user exit processing, see *LE/VSE Programming Guide*.
- For more information about abend codes, see *LE/VSE Programming Guide*.
- For a list of abend code values and reason code values, see *LE/VSE Programming Guide*.

AIXBLD | NOAIXBLD (COBOL Only)

AIXBLD invokes the access method services (AMS) for VSAM key-sequenced data sets (KSDS) and relative-record data sets (RRDS) to complete the file and index definition procedures for COBOL routines.

AIXBLD conforms to the ANSI 1985 COBOL standard.

IBM-Supplied Default: NOAIXBLD

Syntax

NOAIXBLD | NOAIX
Does not invoke the access method services for VSAM key-sequenced and relative-record data sets. NOAIXBLD can be abbreviated to NOAIX.

AIXBLD | AIX
Invokes the access method services for VSAM key-sequenced and relative-record data sets. AIXBLD can be abbreviated to AIX.

Usage Notes

- CICS consideration—This option is ignored under CICS.
- VSE consideration—Access method services messages are directed to the MSGFILE *filename* or, if the file identified by *filename* is not available, to SYSLST.
- The only valid abbreviations for AIXBLD and NOAIXBLD are AIX and NOAIX, respectively.
- When specifying this option in CEEDOPT or CEEUOPT, use the CEEXOPT macro syntax; for example, AIXBLD=(ON) or AIXBLD=(OFF).
- **Use AIXBLD and NOAIXBLD only within a PARM string override**.

Performance Considerations

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

For More Information

- For more information about AIXBLD, see *IBM COBOL for VSE/ESA Programming Guide*
- For more information about the MSGFILE run-time option, see "MSGFILE" on page 33.
- For more information about CEEDOPT, see *LE/VSE Customization Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

ALL31

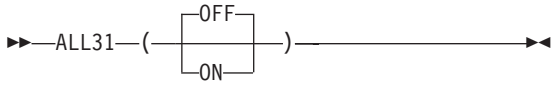
ALL31 specifies whether an application can run entirely in AMODE 31 or whether the application has one or more AMODE 24 routines.

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

ALL31 should have the same setting for all enclaves in the process, because LE/VSE does not support the invocation of a nested enclave requiring ALL31(OFF) from an enclave running with ALL31(ON).

IBM-Supplied Default: ALL31(OFF)

Syntax



OFF

Indicates that one or more routines of an LE/VSE application are AMODE 24.

With ALL31(OFF) specified:

- AMODE switching across calls to LE/VSE common run-time routines is performed. For example, AMODE switching is performed on calls to LE/VSE callable services.
- In COBOL, EXTERNAL data is allocated in storage below the 16MB line.

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

ON

Indicates that no user routines of an LE/VSE application are AMODE 24.

With ALL31(ON) specified:

- AMODE switching across calls to LE/VSE common run-time routines is minimized. For example, no AMODE switching is performed on calls to LE/VSE callable services.
- In COBOL, EXTERNAL data is allocated in unrestricted storage.

Usage Notes

- CICS consideration—The default under CICS is ALL31(ON).
- COBOL consideration—When you link-edit a COBOL program compiled with the NORENT compiler option, the default addressing mode of the link-edited phase is AMODE(ANY). This might result in your program being invoked in 24-bit addressing mode. In order to specify ALL31(ON), your program must be invoked in 31-bit addressing mode. Therefore, you should link-edit your application as AMODE(31). You can use the MODE linkage editor control statement to override the default addressing mode.

Performance Consideration

If your application consists entirely of AMODE (31) routines, it might run faster with ALL31(ON) than with ALL31(OFF) because mode switching code is not required.

For More Information

- For more information about the STACK run-time option, see "STACK" on page 40.

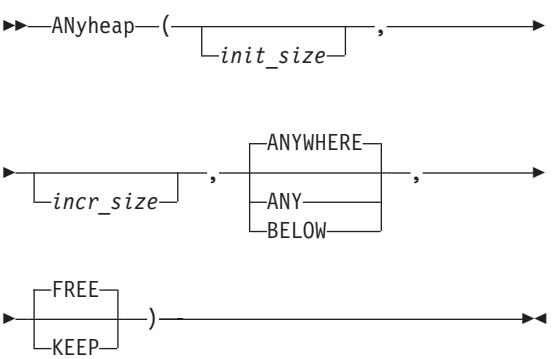
ANYHEAP

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

The ANYHEAP option is always in effect. If you do not specify ANYHEAP or if you specify ANYHEAP(0), LE/VSE allocates the IBM-supplied default value of 16K when a call is made to obtain heap storage.

IBM-Supplied Default:
ANYHEAP(16K,8K,ANYWHERE,FREE)

Syntax



init_size
Determines the minimum initial size of the anywhere 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 anywhere heap area, 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.

ANYWHERE | ANY
Specifies that heap storage can be allocated anywhere in storage. On systems that support bimodal addressing, storage can be allocated either above or below the 16MB line. If there is no storage available above the line, storage is acquired below the line.

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

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

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

Usage Notes

- CICS consideration—Under CICS, the IBM-supplied default is ANYHEAP(4K,4080,ANYWHERE,FREE). Both the initial size and the increment size are rounded up to the nearest multiple of 8 bytes. The minimum is 4K bytes for initial size, and 4080 bytes for increment size..

  Under CICS/VSE 2.3, if ANYHEAP(,,BELOW) is in effect, the maximum initial and increment size for ANYHEAP is 65,504 bytes. If ANYHEAP(,,ANYWHERE) is in effect, the maximum initial and increment size for ANYHEAP is 1 gigabyte (1024M).
- CEEUOPT consideration—If you specify the ANYHEAP run-time option in CEEUOPT, the following default values are used for omitted suboptions:

init_size	32K
incr_size	16K

Performance Considerations

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

For More Information

- For more information about LE/VSE heap storage, see *LE/VSE Programming Guide*.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 38.
- For more information about using the storage report generated by the RPTSTG run-time option to tune your application, see *LE/VSE Programming Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

ARGPARSE | NOARGPARSE (C Only)

ARGPARSE specifies whether command line arguments to a C program are to be parsed. This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive.

Note: You cannot set ARGPARSE as an installation-wide default.

IBM-Supplied Default: ARGPARSE

Syntax

ARGPARSE
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 2 value, and `argv[1]` contains a pointer to the command line string.

Usage Note

- CICS consideration—This option is ignored under CICS.
- You must specify (or use the default) ARGPARSE in order for the REDIR run-time option to have an effect.

For More Information

- For a description of the REDIR run-time option, see "REDIR | NOREDIR (C Only)" on page 36.

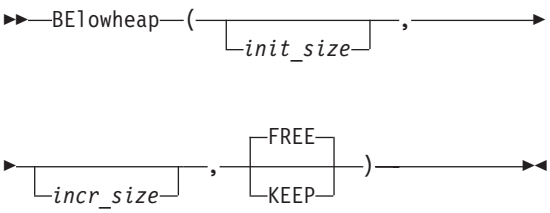
BELOWHEAP

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

The BELOWHEAP option is always in effect. If you do not specify BELOWHEAP or if you specify BELOWHEAP(0), the IBM-supplied default value of 8K is allocated when a call is made to obtain heap storage.

IBM-Supplied Default: BELOWHEAP(8K,4K,FREE)

Syntax



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 16MB 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 is freed.

Usage Notes

- CICS considerations—Under CICS, the IBM-supplied default is BELOWHEAP(4K,4080,FREE). Both the initial size and the increment size are rounded to the nearest multiple of 8 bytes. The minimum is 4K bytes for initial size, and 4080 bytes for increment size. The maximum initial and increment size for BELOWHEAP under CICS/VSE 2.3 is 65,504 bytes.

- CEEUOPT consideration—If you specify the BELOWHEAP run-time option in CEEUOPT, the following default values are used for omitted suboptions:

  init_size 32K

  incr_size 16K

Performance Considerations

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

For More Information

- For more information about LE/VSE heap storage, see *LE/VSE Programming Guide*.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 38.
- For more information about tuning your application, see *LE/VSE Programming Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

CBLOPTS (COBOL Only)

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

You can specify this option only in CEEUOPT or CEEDOPT at initialization.

When you specify the ON suboption of CBLOPTS in CEEUOPT or CEEDOPT, the run-time arguments and program arguments specified in the JCL are honored in the following order:

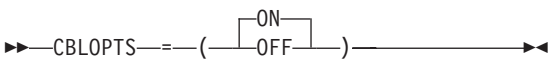
```
program arguments/run-time options
```

This order is the reverse of that normally honored by LE/VSE.

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

IBM-Supplied Default: CBLOPTS=(ON)

Syntax



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

OFF
Specifies that run-time options appear first in the parameter string.

For More Information

- For more information about CEEDOPT, see *LE/VSE Customization Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

CBLPSHPOP (COBOL Only)

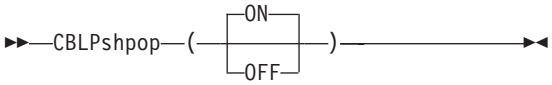
CBLPSHPOP controls whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL (VS COBOL II or COBOL/VSE) subroutine is called.

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

You can set the CBLPSHPOP run-time option on a transaction by transaction basis using CEEUOPT.

IBM-Supplied Default for CICS: CBLPSHPOP(ON)

Syntax



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.

Batch Consideration

The default under BATCH is `CBLPSHPOP=(OFF)`, as this option does not apply to the BATCH environment.

Performance Consideration

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

For More Information

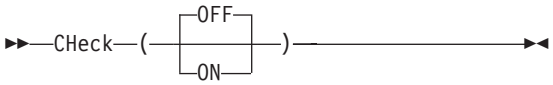
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

CHECK (COBOL Only)

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

IBM-Supplied Default: CHECK(OFF)

Syntax



OFF
Specifies that run-time checking is not performed.

ON
Specifies that run-time checking is performed.

Usage Note

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

Performance Consideration

- Please be aware that CHECK(ON) is required to ensure that the COBOL Compile option SSRANGE takes effect. This option should be used for debugging purposes, and provides COBOL index, subscript, and reference modification range checking.
- If your COBOL program was compiled with SSRANGE, and you are not testing or debugging an application, performance improves when you specify CHECK(OFF).

COUNTRY

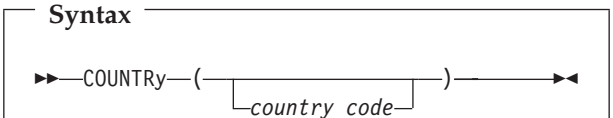
COUNTRY sets the country code, which affects the date and time formats, the currency symbol, the decimal separator, and the thousands separator, based on a specified country. COUNTRY does not change the default settings for the language currency symbol, decimal point, thousands separator, and date and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

You can set the country value using the run-time option COUNTRY or the callable service CEE5CTY.

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

IBM-Supplied Default: COUNTRY(US) with US signifying the United States.

Syntax



country_code
A 2-character code that indicates to LE/VSE the country on which to base the default settings.

Usage Notes

- If you specify a *country_code* that is not available on your system, LE/VSE accepts the value, issues informational message CEE3616I, and uses the default settings listed at the end of Table 32 in Appendix A, "IBM-Supplied Country/Region Code Defaults," on page 227. This is not the same as the installation-supplied default US country code.

CEEUOPT and CEEDOPT permit the specification of an unavailable country code, but give a return code of 4 and a warning message.

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

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

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

For More Information

- For a list of countries and their codes, see Appendix A, "IBM-Supplied Country/Region Code Defaults," on page 227.
- For more information about the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.
- For more about the RPTOPTS and RPTSTG run-time options, see "RPTOPTS" on page 37 and "RPTSTG" on page 38.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Programming Guide*.
- For more information about CEEDOPT, see *LE/VSE Customization Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

DEBUG | NODEBUG (COBOL Only)

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

IBM-Supplied Default: NODEBUG

Syntax

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

- When specifying this option in CEEDOPT or CEEUOPT, use the CEEXOPT macro syntax; for example, DEBUG=(ON) or DEBUG=(OFF).
- **Use DEBUG and NODEBUG only within a PARM string override**.

Performance Consideration

Because DEBUG(ON) gives worse run-time performance than DEBUG(OFF), you should use it only during application development or debugging.

For More Information

- For more details on the USE FOR DEBUGGING declarative, see *IBM COBOL for VSE/ESA Programming Guide*
- For more information about CEEDOPT, see *LE/VSE Customization Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

DEPTHCONDLMT

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

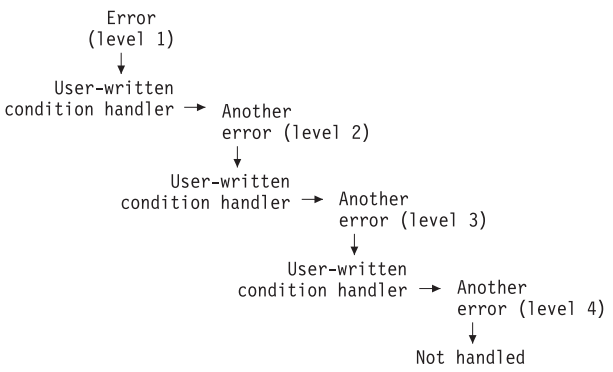
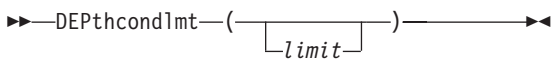


Figure 1. Effect of DEPTHCONDLMT(3) on Condition Handling

IBM-Supplied Default: DEPTHCONDLMT(10)

Syntax



limit

An integer of 0 or greater value. It is the depth of condition handling allowed. An unlimited depth of condition handling is allowed if you specify 0.

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

If the number of nested conditions exceeds the limit, the application terminates with abend 4091 and reason code 21 (X'15').

Usage Notes

- PL/I consideration—DEPTHCONDLMT(0) provides compatibility with previous releases of the DOS PL/I Optimizing Compiler.

For More Information

- For more information on nested conditions, see *LE/VSE Programming Guide*.

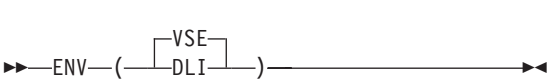
ENV (C Only)

ENV specifies the operating environment for your C application. This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive.

Note: You cannot set ENV as an installation-wide default.

IBM-Supplied Default: ENV(VSE)

Syntax



VSE

Specifies that the C application runs in the VSE batch environment without DL/I.

DLI

Specifies that the C application runs in the VSE batch environment with DL/I.

Usage Note

- CICS consideration—This option is ignored under CICS.

ENVAR (C only)

ENVAR sets the initial values for the environment variables specified in *string*. With ENVAR, you can pass into the application switches or tagged information that can then be accessed using the C functions `getenv`, `setenv`, and `clearenv`.

When the run-time options are merged, ENVAR strings are appended in the order encountered during the merge. Thus, the set of environment variables established by the end of run-time option processing reflects all the various sources where environment variables are specified (rather than just the one source with the highest precedence). However, if a setting for the same environment variable is specified in more than one source, the last setting is used.

Environment variables in effect at the time of the `system` function are copied to the new environment. The copied environment variables are treated the same as those found in the ENVAR run-time option on the command level,

with respect to the merge of the run-time options from their various sources.

When you have specified the RPTOPTS run-time option, you receive a list of the merged ENVAR run-time options. The output for the ENVAR run-time options contains a separate entry for each source where ENVAR was specified with the environment variables from that source.

IBM-Supplied Default: ENVAR(")

Syntax

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

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

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

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

Usage Notes

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

  HLLs can access the environment variables through standard C functions at enclave initialization and throughout the application's run. Access remains until the HLL returns from enclave termination.

For More Information

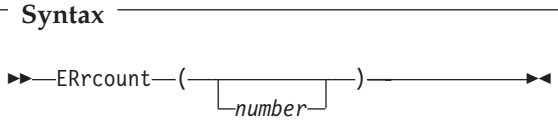
- For more information about the RPTOPTS run-time option, see "RPTOPTS" on page 37.
- For more information about `getenv`, `setenv`, and `clearenv`, see *LE/VSE C Run-Time Programming Guide*.

ERRCOUNT

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

IBM-Supplied Default: ERRCOUNT(20)

Syntax



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

Usage Notes

- ERRCOUNT(0) means the number of conditions that can occur is unlimited. This setting can cause an infinite loop or a runaway task.
- COBOL consideration—LE/VSE counts severity 1 messages with the facility ID IGZ. When the limit is reached, additional severity 1 messages are suppressed.
- PL/I consideration—You should use ERRCOUNT(0) if you are using PL/I.

For More Information

- For more information about the CEEHDLR callable service, see "CEEHDLR—Register User-Written Condition Handler" on page 146.
- For more information about facility IDs, see *LE/VSE Programming Guide*.

EXECOPS | NOEXECOPS (C Only)

EXECOPS specifies whether you can enter run-time options on the command line you use to invoke your application (such as the JCL EXEC statement or the parameter string of the C `system()` function). This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive.

Note: You cannot set EXECOPS as an installation-wide default.

EXECOPS | NOEXECOPS can affect the format of the inbound argument list passed to your application. If you specify NOEXECOPS, the slash (/) that usually separates arguments from run-time options (along with the entire command string) is passed as an argument to the application.

IBM-Supplied Default: EXECOPS

Syntax

EXECOPS
Specifies that you can enter run-time options on the command line.

NOEXECOPS
Specifies that you cannot enter run-time options on the command line. LE/VSE interprets any options on the command line as arguments to the application.

Usage Notes

- CICS consideration—This option is ignored under CICS.

For More Information

- See *LE/VSE Programming Guide* for more information about the argument list format.

HEAP

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

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

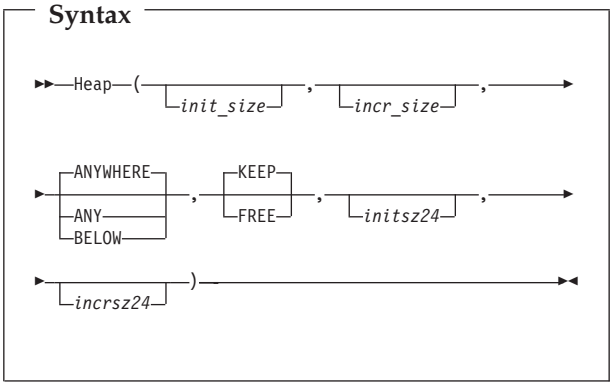
- C variables allocated as a result of the `malloc()`, `calloc()`, and `realloc()` functions
- COBOL WORKING-STORAGE data items
- PL/I variables with the storage class CONTROLLED, or the storage class BASED

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

LE/VSE does not allocate heap storage until the first call to obtain heap storage is made. You can obtain heap storage by using language constructs or by making a call to CEEGTST.

IBM-Supplied Default:
HEAP(32K,32K,ANYWHERE,KEEP,8K,4K)

Syntax



init_size
Determines the minimum initial allocation of heap storage. Specify this 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.

incr_size
Determines the minimum size of any subsequent increment to the heap storage. Specify this 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.

ANYWHERE | ANY
Specifies that you can allocate heap storage anywhere in storage. On systems that support bimodal addressing, you can allocate storage either above or below the 16MB line. If there is no available storage above the line, storage is acquired below the line. On systems that do not support bimodal addressing (for example, when VSE/ESA is running on a 370-mode machine, or as a 370-mode guest system under VM), LE/VSE ignores this option and places heap storage below 16MB.

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

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

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

initsz24
Determines the minimum initial size of the heap storage that is obtained below the 16MB line for applications running with ALL31(OFF) when these applications specify ANYWHERE in the HEAP run-time option. Specify *initsz24* as *n*, *n*K, or *n*M number of bytes. The amount of storage is rounded up to the nearest multiple of 8 bytes.

initsz24 applies to all heaps that are not allocated strictly below the 16MB line.

incrsz24
Determines the minimum size of any subsequent increment to the heap area that is obtained below the 16MB line for applications running with ALL31(OFF) when these applications specify ANYWHERE in the HEAP run-time option. Specify *incrsz24* as *n*, *n*K, or *n*M number of bytes. The amount of storage is rounded up to the nearest multiple of 8 bytes.

incrsz24 applies to all heaps that are not allocated strictly below the 16MB line.

Usage Notes

- Applications running in AMODE 24 that request heap storage get the storage below the 16MB line regardless of the setting of ANYWHERE | BELOW.
- COBOL consideration—You can use the HEAP option to provide some of the function provided by 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(,,ANYWHERE) is in effect.
  - The main routine is AMODE 31.
- CICS consideration—The IBM-supplied default is HEAP(4K,4080,ANYWHERE,KEEP,4K,4080). Both the initial HEAP allocation and HEAP increments are rounded to the next higher multiple of 8 bytes. The minimum is 4K bytes for initial size, and 4080 bytes for increment size..

  Under CICS/VSE 2.3, if HEAP(,,BELOW) is in effect, the maximum size of a heap segment is 65,504 bytes. If too large a value is specified, the application fails at the first attempt to allocate heap storage. If HEAP(,,ANYWHERE) is in effect, the maximum size of a heap segment is 1 gigabyte (1024M). These restrictions are subject to change from one release of CICS to another.
- CEEUOPT consideration—If you specify the HEAP run-time option in CEEUOPT, the following default values are used for omitted suboptions:

init_size	64K
incr_size	64K
initsz24	16K
incrsz24	16K

Performance Considerations

The RPTSTG run-time option generates a report of the storage your application uses while running. To improve performance, use the information in the storage report generated by the RPTSTG run-time option as an aid in setting the initial and increment size for HEAP.

For More Information

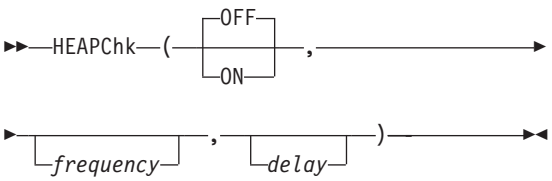
- For more information about LE/VSE heap storage, see *LE/VSE Programming Guide*
- For more information about the CEECRHP and CEEGTST callable services, see "CEECRHP—Create New Additional Heap" on page 103 and "CEEGTST—Get Heap Storage" on page 143.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 38.
- For more information about using the storage report generated by the RPTSTG run-time option to tune your application, see *LE/VSE Programming Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

HEAPCHK

HEAPCHK provides a checking facility to verify that the heap storage has not been damaged.

IBM-Supplied Default: HEAPCHK(OFF,1,0)

Syntax



OFF
Specifies that no heap checking will be done.

ON
Specifies that heap checking will be activated and controlled by the *frequency* and *delay* parameters.

frequency
Determines the event frequency at which heap checking is to occur. This specifies that heap storage will be checked for damage on every *n*th call to an LE/VSE storage management service. Specify this value as *n*, *n*K, or *n*M.

delay
Determines the number of calls to LE/VSE storage management services that will be made before activating the heap checking mechanism. Specify this value as *n*, *n*K, or *n*M.

Usage Notes

- When specifying values for *frequency* and *delay*, remember that storage management services are called by LE/VSE's internal routines in addition to your application calls.
- Certain language constructs will also call LE/VSE storage management services. For example, PL/I ALLOCATE and FREE statements for variables and aggregates that are not within a PL/I AREA, and the C `malloc()` and `free()` library functions.
- EXEC CICS GETMAIN and FREEMAIN do not use LE/VSE storage management services.

Performance Considerations

HEAPCHK is intended to be used in a test environment only!. Use HEAPCHK in production only when necessary, as it will use extra CPU resources and degrade performance.

For More Information

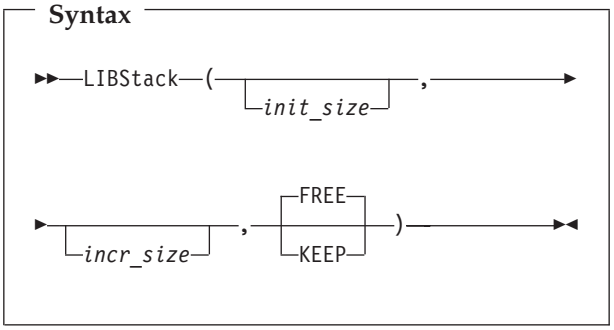
For more information about LE/VSE's storage management services, see:
- "CEEGTST—Get Heap Storage" on page 143
- "CEEFRST—Free Heap Storage" on page 129

LIBSTACK

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

IBM-Supplied Default:
LIBSTACK(12K,4K,FREE)

Syntax



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

Specify *init_size* as *n*, *n*K, or *n*M bytes of storage. *init_size* can be preceded by a minus sign. In the batch environment, if you specify a negative number, all available storage minus the amount specified is used for the initial stack segment.

In the batch environment, an *init_size* of 0 or −0 requests half of the largest block of contiguous storage below the 16MB line.

At initialization, LE/VSE allocates the storage rounded up to the nearest multiple of 8 bytes.

incr_size
Determines the minimum size of any subsequent increment to the library stack area. Specify this value as *n*, *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 do not specify *incr_size*, LE/VSE uses the IBM-supplied default setting of 4K. If *incr_size*=0, LE/VSE obtains only the amount of storage needed at the time of the request, rounded up to the nearest multiple of 8 bytes.

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

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

KEEP
Specifies that LE/VSE does not release storage allocated to LIBSTACK increments when the last of the storage is freed.

Usage Notes

- CICS consideration— The IBM-supplied default setting for LIBSTACK is LIBSTACK(4K,4080,FREE). Both the initial and increment sizes are rounded up to the next multiple of 8 bytes. The minimum is 4K bytes for initial size, and 4080 bytes for increment size.

  Under CICS, the maximum initial and increment size for LIBSTACK is 65,504 bytes.
- CEEUOPT consideration—If you specify the LIBSTACK run-time option in CEEUOPT, the following default values are used for omitted suboptions:

init_size	32K
incr_size	16K

Performance Considerations

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

For More Information

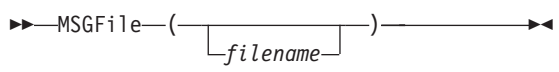
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 38.
- For more information about using the storage report generated by the RPTSTG run-time option to tune your application, see *LE/VSE Programming Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

MSGFILE

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

IBM-Supplied Default: MSGFILE(SYSLST)

Syntax



filename
The filename of the run-time diagnostics file.

Usage Notes

- CICS considerations—The MSGFILE option defaults to the CESE transient data queue:

`MSGFILE=(CESE)`. Specification of a different transient data queue for MSGFILE is possible. Though it is the users responsibility to ensure that this transient data queue is available in the CICS system. If an option other than CESE is specified for MSGFILE and this transient data queue becomes unusable or unavailable, LE/VSE will default to the CESE transient data queue. The CESE transient data queue must always be available either as TYPE=INDIRECT or TYPE=EXTRA in the CICS DCT definition. The MSGFILE destination name under CICS must not exceed 4 characters in length. Truncation will occur on the MSGFILE destination if the name used is greater than 4 characters in length. The supplied definition of CEEMSG in CEECDCT.A in PRD2.SCEEBASE should be used as an example for any other TYPE=SDSCI destinations being used as a Disk File destination for MSGFILE.
Note that if a DISK file is being used as a final destination, you must remember to add 8 bytes to the BLKSIZE specified in your DCT definition. Any MSGFILE destination used must support a blksize of at least 175 bytes (inclusive of the 8 bytes required for LIOCS output files if DISK is used). The VSE system console is not a supported destination for MSGFILE either directly or indirectly.

- HLL compile-time options can affect whether your run-time output goes to MSGFILE *filename*.
- LE/VSE does not check the validity of the MSGFILE *filename*. An invalid *filename* generates an error condition on the first attempt to issue a message.
- C consideration—C `perror()` messages and output directed to `stderr` go to the MSGFILE destination.
- PL/I consideration—Run-time messages in PL/I routines are directed to the file specified by the LE/VSE MSGFILE run-time option, instead of to the PL/I SYSPRINT STREAM PRINT file.

  User-specified output is still directed to the PL/I SYSPRINT STREAM PRINT file. If you want LE/VSE to handle this output, specify MSGFILE(SYSPRINT). When you specify MSGFILE(SYSPRINT), all PL/I run-time messages and user-specified output are directed to SYSLST.

For More Information

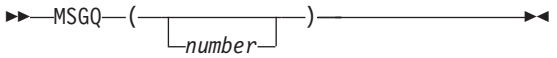
- For more information about the RPTOPTS and RPTSTG run-time options, see “RPTOPTS” on page 37 and “RPTSTG” on page 38.
- For more information about the CEEMSG and CEEMOUT callable services, see “CEEMSG—Get, Format, and Dispatch a Message” on page 171 and “CEEMOUT—Dispatch a Message” on page 162.
- For details on how HLL compiler options affect messages, see information on HLL I/O statements and message handling in *LE/VSE Programming Guide*.
- For examples of getting and formatting messages using LE/VSE-conforming languages, see “CEEMSG—Get, Format, and Dispatch a Message” on page 171.
- For more information about the CESE transient data queue, see *LE/VSE Programming Guide*.
- For more information about `perror()` and `stderr`, see C message output information in *LE/VSE Programming Guide*.

MSGQ

MSGQ specifies the number of ISI blocks that LE/VSE allocates on a per thread basis for use by the application. The ISI contains information that LE/VSE uses to identify and react to conditions, provide access to q_data tokens, and assign space for message inserts used with user-created messages. When an ISI is needed and one is not available, LE/VSE takes the least recently used ISI for reuse. CEECMI allocates storage for the ISI, if necessary.

IBM-Supplied Default: MSGQ(15)

Syntax



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

For More Information

- For more information about the CEECMI callable service, see "CEECMI—Store and Load Message Insert Data" on page 101.
- For more information about the ISI, see *LE/VSE Programming Guide*.

NATLANG

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

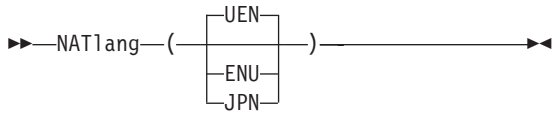
NATLANG affects only the LE/VSE NLS and date and time services, not the LE/VSE locale callable services.

You can set the national language by using the NATLANG run-time option or the SET option of the CEE5LNG callable service. LE/VSE maintains one current language at the enclave level. This current language remains in effect until one of the above changes it. For example, if you specify JPN in the NATLANG run-time option, but subsequently specify ENU using the CEE5LNG callable service, ENU becomes the current national language.

LE/VSE writes certain parts of storage and options reports and dump output only in mixed-case U.S. English, and certain abnormal termination messages only in uppercase U.S. English.

IBM-Supplied Default: NATLANG(UEN)

Syntax



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

Message text consists of SBCS (single-byte character set) characters and includes only uppercase letters.

ENU
A 3-character ID specifying mixed-case U.S. English.

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

JPN
A 3-character ID specifying Japanese.

Message text can contain a mixture of SBCS and DBCS (double-byte character set) characters.

Usage Notes

- If you specify a national language that is not available on your system, LE/VSE uses the IBM-supplied default UEN (uppercase U.S. English).

  CEEUOPT and CEEDOPT can specify an unknown national language code, but give a return code of 4 and a warning message.
- C consideration—LE/VSE provides locales used in 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 run-time option. NATLANG affects only LE/VSE NLS and date and time services. `setlocale()` and CEESETL affect only C locale-sensitive functions and LE/VSE locale callable services.

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

For More Information

- For more information about the CEE5LNG and CEESETL callable services, see "CEE5LNG—Set National Language" on page 79 and "CEESETL—Set Locale Operating Environment" on page 191.
- For more information about `setlocale()`, see *LE/VSE C Run-Time Programming Guide*.
- For more information about CEEDOPT, see *LE/VSE Customization Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

PLIST (C Only)

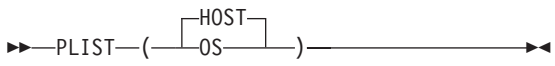
PLIST specifies the format of the invocation parameters your C application receives when you invoke it.

This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive.

Note: You cannot set PLIST as an installation-wide default.

IBM-Supplied Default: PLIST(HOST)

Syntax



HOST
The parameter list is a character string. The string received by your C application is assumed to be in a VSE format, using a string prefixed by a halfword length.

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

Usage Notes

- CICS consideration—This option is ignored under CICS.
- DL/I consideration—Specify PLIST(OS) when your application runs in the batch environment with DL/I.

REDIR | NOREDIR (C Only)

REDIR specifies whether you can enter redirections for `stdin`, `stderr`, and `stdout` from the command line.

This option does not apply to non-C languages and can be specified only with the C `#pragma runopts` directive.

Note: You cannot set REDIR as an installation-wide default.

For ILC applications, REDIR can be used to direct `printf` output from the application to the MSGFILE so it can be interspersed with output from COBOL programs that use the compiler option OUTDD, or output from PL/I routines that use the MSGFILE(SYSPRINT) run-time option.

IBM-Supplied Default: REDIR

Syntax

REDIR
Specifies that you can redirect `stdin`, `stderr`, and `stdout` from the command line.

REDIR applies only if ARGPARSE is also specified or defaulted.

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

Usage Notes

- CICS consideration—This option is ignored under CICS.

For More Information

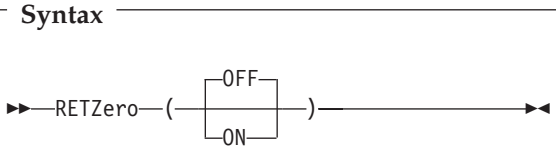
- For more information about the MSGFILE run-time option, see "MSGFILE" on page 33.
- For details about `printf`, see *LE/VSE Programming Guide*.
- For more information about the ARGPARSE run-time option, see "ARGPARSE | NOARGPARSE (C Only)" on page 24.

RETZERO (COBOL Only)

RETZERO will ensure that, if the run unit does not abend or terminate abnormally, the user return code will be set to zero regardless of the contents of register 15 or the RETURN-CODE special register.

IBM-Supplied Default: RETZERO(OFF)

Syntax



OFF
Specifies that the user return code will be unchanged.

ON
Specifies that LE/VSE will set the user return code to zero before terminating the run unit.

Usage Notes

This option is intended for COBOL programs that call Assembler subroutines, where the subroutine does not clear register 15 before returning to the calling program. Under normal circumstances (with RETZERO(OFF)), the contents of register 15 will become the contents of the RETURN-CODE special register, and if the COBOL program does not subsequently change this, it will be used as the user return code for the enclave. As this is an unpredictable value (possibly a virtual storage address), it can cause errors in the batch job stream. With RETZERO(ON), the user return code will be forced to zero before the run unit ends.

RPTOPTS

RPTOPTS generates, after an application has run, a report of the run-time options in effect while the application was running. LE/VSE writes options reports only in mixed-case U.S. English.

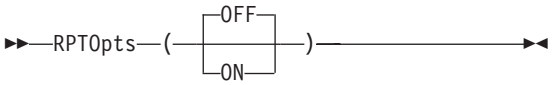
In the batch environment, LE/VSE directs the report to the *filename* specified in the MSGFILE run-time option. In the CICS environment, LE/VSE directs the report to the CESE transient data queue.

Figure 2 on page 38 shows the sample output when RPTOPTS is set to ON. RPTOPTS(ON) lists the declared run-time options in alphabetic order. The report lists the option names and shows where each option obtained its current setting. The report heading displayed at the top of the options report is set by CEE5RPH. The date and time formats are affected by the country code set by the COUNTRY run-time option or the CEE5CTY callable service.

The LAST WHERE SET column in the report shows the last place where the options were referenced, even if no suboptions or subsets of the options were changed. “Programmer default” includes any options specified with C `#pragma runopts`, PL/I PLIXOPT, and CEEUOPT.

IBM-Supplied Default: RPTOPTS(OFF)

Syntax



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

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

Performance Considerations

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

```
Options Report for Enclave CBLDATE 06/01/04 10:06:17 AM
Language Environment for VSE/ESA V1 R4.4

LAST WHERE SET          OPTION
------------------------------------------------------------------------
Programmer default        ABPERC(NONE)
Installation default      ABTERMENC(ABEND)
Installation default    NOAIXBLD
Programmer default        ALL31(OFF)
Programmer default        ANYHEAP(16384,8192,ANYWHERE,FREE)
Installation default      BELOWHEAP(8192,4096,FREE)
Installation default      CBLOPTS(ON)
Installation default      CBLPSHPOP(OFF)
Installation default      CHECK(OFF)
Non-overrideable          COUNTRY(US)
Installation default    NODEBUG
Installation default      DEPTHCONDLMT(10)
Installation default      ENVAR("")
Programmer default        ERRCOUNT(20)
Installation default      HEAP(32768,32768,ANYWHERE,KEEP,8192,4096)
Installation default      HEAPCHK(OFF,1,0)
Installation default      LIBSTACK(12288,4096,FREE)
Installation default      MSGFILE(SYSLST)
Installation default      MSGQ(15)
Installation default      NATLANG(UEN)
Installation default      RETZERO(OFF)
Invocation command        RPTOPTS(ON)
Installation default      RPTSTG(OFF)
Installation default    NORTEREUS
Installation default      STACK(131072,131072,BELOW,KEEP)
Assembler user exit       STORAGE(00,NONE,NONE,32768)
Programmer default        TERMTHDACT(DUMP,LSTQ,0)
Installation default    NOTEST(ALL,"*","PROMPT","")
Installation default      TRACE(OFF,4096,DUMP,LE=0)
Installation default      TRAP(ON,MAX)
Installation default      UPSI(00000000)
Installation default    NOUSRHDLR()
Programmer default        XUFLOW(AUTO)

LSTQ Options Report
------------------------------------------------------------------------
LSTQ Class Setting         L
LSTQ Disposition Setting   D
LSTQ Remote Node ID        LETEST
LSTQ Remote User ID        LETEST
```

Figure 2. Options Report Produced by LE/VSE Run-Time Option RPTOPTS(ON)

When TERMTHDACT is set with the LSTQ suboption, a "LSTQ Options Report" will be produced. This report shows the LSTQ options that are currently active. You can change these settings using the CEELOPT macro. This macro is supplied with the CEECOPT and CEEDOPT samples, which are used to set the LE/VSE default environment-wide run-time options.

The "LSTQ Options Report" shown at the end of Figure 2 was produced using the settings TERMTHDACT(DUMP,LSTQ,0).

For More Information

- For more information about the MSGFILE and COUNTRY run-time options, see "MSGFILE" on page 33 and "COUNTRY" on page 26.
- For more information about the CEE5RPH and CEE5CTY callable services, see "CEE5RPH—Set Report Heading" on page 89 and "CEE5CTY—Set Default Country" on page 65.
- For more information about the CEECOPT sample, see "Changing the Installation-Wide Run-Time Options Default (CICS)" in the *LE/VSE Customization Guide*.
- For more information about the CEEDOPT sample, see "Changing the Installation-Wide Run-Time Options Default (BATCH)" in the *LE/VSE Customization Guide*.

RPTSTG

RPTSTG generates, after an application has run, a report of the storage the application used. In the batch environment, the report is directed to the *filename* specified in the MSGFILE run-time option. In the CICS environment, the report is directed to the CESE transient data queue.

Figure 3 shows a sample report created with the RPTSTG option set to ON.

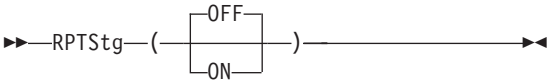
The storage report heading is set by CEE5RPH. The date and time formats, in the RPTSTG generated reports, are affected by the country code set by the COUNTRY run-time option or the CEE5CTY callable service.

You can use the storage report information to adjust the ANYHEAP, BELOWHEAP, HEAP, LIBSTACK, and STACK run-time options.

LE/VSE writes storage reports only in mixed-case U.S. English.

IBM-Supplied Default: RPTSTG(OFF)

Syntax



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.

Usage Notes

The statistics for initial and incremental allocations of storage types that have a

corresponding run-time option, differ from the run-time option settings when (1) their values have been rounded up by the implementation, or (2) when allocations larger than the amounts specified were required during execution. All of the following are rounded up to an integral number of double words:

- Initial STACK allocations
- Initial allocations of all types of Heap
- Incremental allocations of all types of stack and heap

The phrases "Number of segments allocated" and "Number of segments freed" represent the following:

- In the batch environment, the number of GETVIS and FREEVIS requests, respectively.
- In CICS, the number of EXEC CICS GETMAIN and EXEC CICS FREEMAIN requests, respectively.

Performance Considerations

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

The storage report generated by RPTSTG(ON) shows the number of system-level get storage calls that were required while the application was running.

To improve performance, use the information in the storage report generated by the RPTSTG option as an aid in setting the initial and increment size for STACK and HEAP for applications that need tuning in the area of storage usage. These values can be used as input to an application-specific CEEUOPT options module, and included in the application program during the link edit. They should not be used as values for installation defaults (such as those used in CEECOPT or CEEDOPT). This reduces the number of times that the LE/VSE storage manager makes requests to acquire storage.

```
Storage Report for Enclave CBLDATE 06/01/04 9:16:06 AM
Language Environment for VSE/ESA V1 R4.4

    STACK statistics:
      Initial size:                                    131072
      Increment size:                                  131072
      Total stack storage used (sugg. initial size):     9056
      Number of segments allocated:                         1
      Number of segments freed:                             0
    LIBSTACK statistics:
      Initial size:                                     12288
      Increment size:                                    4096
      Total stack storage used (sugg. initial size):     1144
      Number of segments allocated:                         1
      Number of segments freed:                             0
    HEAP statistics:
      Initial size:                                     32768
      Increment size:                                   32768
      Total heap storage used (sugg. initial size):         0
      Successful Get Heap requests:                         0
      Successful Free Heap requests:                        0
      Number of segments allocated:                         0
      Number of segments freed:                             0
    ANYHEAP statistics:
      Initial size:                                     16384
      Increment size:                                    8192
      Total heap storage used (sugg. initial size):      2480
      Successful Get Heap requests:                        12
      Successful Free Heap requests:                        0
      Number of segments allocated:                         1
      Number of segments freed:                             0
    BELOWHEAP statistics:
      Initial size:                                      8192
      Increment size:                                    4096
      Total heap storage used (sugg. initial size):      1184
      Successful Get Heap requests:                         6
      Successful Free Heap requests:                        3
      Number of segments allocated:                         1
      Number of segments freed:                             0
    Additional Heap statistics:
      Successful Create Heap requests:                      0
      Successful Discard Heap requests:                     0
      Total heap storage used:                              0
      Successful Get Heap requests:                         0
      Successful Free Heap requests:                        0
      Number of segments allocated:                         0
      Number of segments freed:                             0
End of Storage Report
```

Figure 3. Storage Report Produced by LE/VSE Run-Time Option RPTSTG(ON)

For More Information

- For more information about the MSGFILE and COUNTRY run-time options, see "MSGFILE" on page 33 and "COUNTRY" on page 26.
- For more information about the CEE5RPH and CEE5CTY callable services, see "CEE5RPH—Set Report Heading" on page 89 and "CEE5CTY—Set Default Country" on page 65.
- For more information about the BELOWHEAP and HEAP run-time options, see "BELOWHEAP" on page 24 and "HEAP" on page 30.
- For more information about the LIBSTACK and STACK run-time options, see "LIBSTACK" on page 32 and "STACK" on page 40.

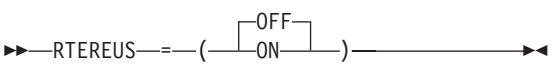
- For more information about using the storage report generated by the RPTSTG run-time option to tune your application, see *LE/VSE Programming Guide*.

RTEREUS | NORTEREUS (COBOL Only)

RTEREUS implicitly initializes the run-time environment to be reusable when the main program for the thread is a COBOL program. This option is valid only when used with CEEDOPT or CEEUOPT.

IBM-Supplied Default: RTEREUS=(OFF)

Syntax



OFF
Does not initialize the run-time environment to be reusable when the first COBOL routine is invoked.

ON
Initializes the run-time environment to be reusable when the first COBOL routine is invoked.

Usage Notes

- Avoid using RTEREUS=(ON) as an installation default, because doing so can cause problems for other HLLs such as C and PL/I.
- The IGZERREO CSECT affects the handling of program checks in the non-Language Environment conforming driver that repeatedly invokes COBOL programs.
- CICS consideration—This option is ignored under CICS.

Performance Considerations

You must change STOP RUN statements to GOBACK statements in order to gain the benefits of RTEREUS. STOP RUN terminates the reusable environment. If you specify RTEREUS, LE/VSE 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.

Notes:

1. The IGZERREO CSECT affects the performance of running with RTEREUS.
2. LE/VSE also offers preinitialization support in addition to RTEREUS.

For More Information

- For more information about CEEDOPT, see *LE/VSE Customization Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.
- For more information about preinitialization, see *LE/VSE Programming Guide*.

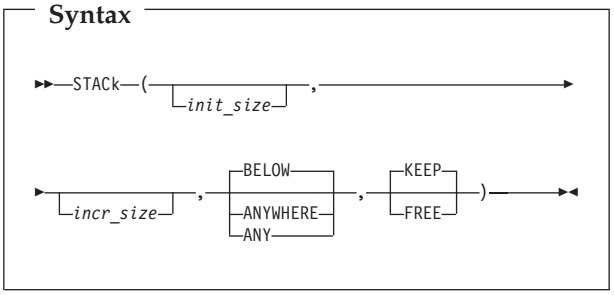
STACK

STACK controls the allocation of the thread's stack storage. Typical items residing in the stack are C or PL/I automatic variables, and temporary work areas for COBOL library routines.

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.

IBM-Supplied Default:
STACK(128K,128K,BELOW,KEEP)

Syntax



init_size
Determines the size of the initial 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 8 bytes.

init_size can be preceded by a minus sign. In the batch environment, if you specify a negative number LE/VSE 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 16MB line. Behavior under CICS is described in the Usage Notes for this run-time option.

incr_size
Determines the minimum size of any subsequent increment to the 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 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 4K, 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, *incr_size* is specified as 8K, and the initial stack segment is full, LE/VSE obtains a 9000-byte stack increment from the operating system to satisfy the request. If the requested size is smaller than 8K, LE/VSE obtains an 8K stack increment from the operating system.

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

ANYWHERE | ANY
Specifies that stack storage can be allocated anywhere in storage. On systems that support bimodal addressing, storage can be allocated either above or below the 16MB line. If there is no storage available above the line, LE/VSE acquires storage below the line.

The only valid abbreviation for ANYWHERE is ANY.

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.

Usage Notes

- Applications running with ALL31(OFF) must specify STACK(,,BELOW) to ensure that stack storage is addressable by the application.
- CICS consideration—The IBM-supplied default setting for STACK under CICS is STACK(4K,4080,ANYWHERE,KEEP).

  Under CICS, the maximum initial and increment size for STACK below 16MB is 65,504 bytes. The maximum initial and increment size for STACK above 16MB is 1 gigabyte (1024M). This restriction is subject to change from one release of CICS to another.

  Both the initial size and the increment size are rounded up to the nearest multiple of 8 bytes. The initial size minimum is 4K bytes, the increment size minimum is 4080 bytes.

  If you do not specify STACK, LE/VSE assumes the default value of 4K. Under CICS, STACK(0), STACK (–0), and STACK (–n) are all interpreted as STACK(4K).
- COBOL consideration—When you linkedit a COBOL program compiled with the NORENT compiler option, the default addressing mode of the linkedited phase is AMODE(ANY). This might result in your program being invoked in 24-bit addressing mode. If you wish to specify STACK(ANY) to obtain better storage utilisation, your program must be invoked in 31-bit addressing mode. Therefore, you might wish to compile your program specifying the RENT option. Alternatively, you can use the MODE linkage editor control statement to override the addressing mode to AMODE(31). **Note:** You are strongly recommended **not** to change the residency mode (RMODE) for programs compiled with the RENT or NORENT options.
- PL/I consideration—PL/I automatic storage above the 16MB line is supported under control of the LE/VSE STACK option. When the LE/VSE stack is above, PL/I temporaries (dummy arguments) and parameter lists (for reentrant/recursive blocks) also reside above.

  The stack frame size for an individual block is constrained to 16MB. Stack frame extensions are also constrained to 16MB. Therefore, the size of an automatic aggregate, temporary variable, or dummy argument cannot exceed 16MB. Violation of this constraint might have unpredictable results.

- CEEUOPT consideration—If you specify the STACK run-time option in CEEUOPT, the following default values are used for omitted suboptions:

  init_size 512K

  incr_size 512K

Performance Considerations

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

For More Information

- For more information about the ALL31 run-time option, see "ALL31" on page 22.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 38.
- For more information about using the storage report generated by the RPTSTG run-time option to tune your application, see *LE/VSE Programming Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

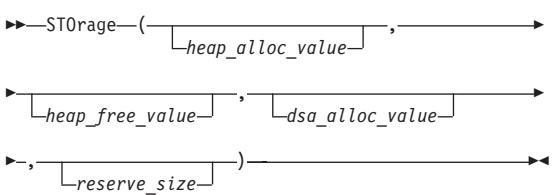
STORAGE

STORAGE controls the initial content of storage when allocated and freed, and the amount of storage that is reserved for the out-of-storage condition. If you specify one of the parameters in the STORAGE run-time option, all allocated storage processed by the parameter is initialized to that 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.

IBM-Supplied Default:
STORAGE(00,NONE,NONE,32K)

Syntax



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 quotes. 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 quotes. 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 the default value of NONE, the allocated heap storage is not initialized.

heap_free_value
The value with which any heap storage freed by the storage manager is overwritten. You can specify *heap_free_value* as:

- A single character enclosed in quotes. For example, a *heap_free_value* of 'f' overwrites freed heap storage to X'868686...86'; 'B' overwrites freed heap storage to X'C2'.
- Two hex digits without quotes. A *heap_free_value* of FE overwrites freed heap storage with X'FEFEFE...FE'.
- NONE. If you specify the default value of NONE, the freed heap storage is not overwritten.

dsa_alloc_value
The initialized value of stack frames from the LE/VSE stack. A stack frame is dynamically-acquired storage that is composed of a standard register save area and the area available for automatic storage.

If specified, all LE/VSE stack storage including automatic variable storage is initialized to *dsa_alloc_value*. Stack frames allocated outside the LE/VSE stack are never initialized.

You can specify *dsa_alloc_value* as:

- A single character enclosed in quotes. If you specify a single character, any dynamically-acquired stack storage allocated by the storage manager is initialized to that character's EBCDIC equivalent. For example, if you specify 'A' as the *dsa_alloc_value*, stack storage is initialized to X'C1'. A *dsa_alloc_value* of 'F' initializes stack storage to X'C6', 'd' to X'84'.
- Two hex digits without quotes. If you specify two hex digits, any dynamically acquired stack storage is initialized to that value. For example, if you specify FE as the *dsa_alloc_value*, stack storage is initialized to X'FE'. A *dsa_alloc_value* of 00 initializes stack storage to X'00', FF to X'FF'.
- NONE. If you specify the default value of NONE, the stack storage is not initialized.

reserve_size

The amount of storage for the LE/VSE 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 up to the nearest multiple of 8 bytes.

If you specify *reserve_size* as 0, no reserve segment is allocated. If you do not specify a reserve segment and your application runs out of storage, the application abends with a return code of 4088 and a reason code of 1004.

If you specify a *reserve_size* that is greater than 0 in the batch environment, LE/VSE does not immediately abend when your application runs out of storage. Instead, when the stack overflows, LE/VSE attempts to get another stack segment and add it to the stack.

If unsuccessful, LE/VSE temporarily adds the reserve stack segment to the overflowing stack, and signals the out-of-storage condition. This allows a user-written condition handler to gain control and release storage. If the reserve stack segment overflows while this is happening, LE/VSE abends with a return code of 4088 and reason code of 1004.

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

Usage Notes

- *heap_alloc_value*, *heap_free_value*, and *dsa_alloc_value* can all be enclosed in quotes. 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 quotes. Both of the following are correct ways to specify a single quote:

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

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

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

- CICS consideration—The IBM-supplied default setting for STORAGE under CICS is STORAGE(00,NONE,NONE,0K).

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

  If a reserved segment size is specified, either as a default or an override under CICS, this storage size will be allocated but never used. This results in wasted 24-bit storage. Your are recommended to always use a reserve segment size of 0k under CICS.

- CEEUOPT consideration—If you specify the STORAGE run-time option in CEEUOPT, the default value 8K is used if the *reserve_size* suboption is omitted.

PL/I for VSE/ESA Users

To provide similar storage initialization as DOS/PL1, use STORAGE=(00,NONE,00,32K) as an installation default setting. Please note that a performance degradation may result from using this setting.

Performance Considerations

Using STORAGE to control initial values can increase program run time. 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.

Use STORAGE(00,NONE,NONE,0K) when you are not debugging.

TERMTHDACT

TERMTHDACT sets the level of information that is produced when LE/VSE percolates a condition of severity 2 or greater beyond the first routine's stack frame.

The UADUMP suboption of TERMTHDACT produces the same output that the DUMP suboption provides, and also produces a:
- system dump of the user partition in batch.
- transaction dump under CICS.

The LE/VSE service CEE5DMP is called for the TRACE, DUMP, and UADUMP suboptions of TERMTHDACT.

The following CEE5DMP options are passed for TRACE:
- CONDITION
- FNAME(CEEDUMP)
- NOBLOCKS
- NOENTRY
- FILES
- NOSTORAGE
- VARIABLES
- PAGESIZE(60)
- STACKFRAME(ALL)
- THREAD(ALL)
- TRACEBACK

The following options are passed for DUMP and UADUMP:

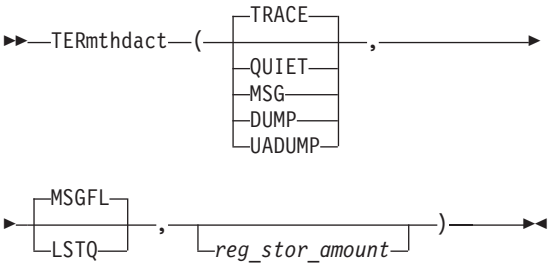
BLOCKS
CONDITION
FILES
FNAME(CEEDUMP)
NOENTRY
PAGESIZE(60)
STACKFRAME(ALL)
STORAGE
THREAD(ALL)
TRACEBACK
VARIABLES

If a message is printed (based upon the TERMTHDACT(MSG) run-time option), the message is for the active condition immediately 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 run-time 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 run-time option.
- Under abnormal termination, the following dump contents are generated:
  - TERMTHDACT(QUIET)—generates a dump containing the trace table only
  - TERMTHDACT(MSG)—generates a dump containing the trace table only
  - TERMTHDACT(TRACE)—generates a dump containing the trace table and the traceback
  - TERMTHDACT(DUMP)—generates a dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block)
  - TERMTHDACT(UADUMP)—generates a dump containing thread/enclave/process storage and control blocks (the trace table is included as an enclave control block). Also produces a system dump of the user partition.
- Under normal termination, the following dump contents are generated:
  - Independent of the TERMTHDACT setting, LE/VSE generates a dump containing the trace table only.

IBM-Supplied Default:
TERMTHDACT(TRACE,,0)

Syntax



TRACE
Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, LE/VSE generates a message indicating the cause of the termination and a trace of the active routines on the activation stack.

QUIET
Specifies that LE/VSE 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, LE/VSE 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, LE/VSE generates a message indicating the cause of the termination, a trace of the active routines on the activation stack, and an LE/VSE dump.

A currently active run-time options report will also be sent to the dump destination for problem diagnosis assistance, even if RPTOPTS(OFF) has been previously specified. If RPTOPTS(ON) has been specified and a dump is produced in response to a failure, a single run-time options report only will be generated within the dump output.

UADUMP
Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, LE/VSE generates a message indicating the cause of the termination, a trace of the active routines on the activation stack, and an LE/VSE dump, and, depending on the JCL OPTION DUMP setting, a system dump of the user partition.

If abnormal termination occurs while TERMTHDACT(UADUMP) is set, if possible LE/VSE will attempt to re-execute the failing instruction, to produce a system dump. If this is not possible, LE/VSE will issue a JDUMP macro to terminate the enclave with a system dump.

A currently active run-time options report will also be sent to the dump destination for problem diagnosis assistance, even if RPTOPTS(OFF) has been previously specified. If RPTOPTS(ON) has been specified and a dump is produced in response to a failure, a single run-time options report only will be generated within the dump output.

MSGFL
Under CICS, causes all dump output to be sent to the output destination specified by the run-time option MSGFILE. For BATCH environments, the destination is dependant upon the presence or absence of a CEEDUMP label. In the BATCH environment, the default destination is the device assigned to SYSLST.

LSTQ
Causes all dump output to be sent to the VSE/POWER LSTQ.

Usage Notes:

- The LSTQ option will only take effect when a condition of severity 2 or greater goes unhandled and a dump is requested.
- Callable services, such as CEE5DMP, will still have dump output sent to the destination specified in the MSGFILE runtime option.
- Abend messages and LE/VSE reports will still be sent to the MSGFILE destination even if LSTQ is set. The exception to this is when DUMP or UADUMP are set and a run-time options report is generated as part of the dump output. In this case, the options report will be send to the LSTQ member along with the dump output.
- A TERMTHDACT level setting of TRACE or more must be in effect for dump output to be produced. Otherwise all output will still be sent to the MSGFILE destination.
- Output sent to the VSE/POWER LSTQ will be stored on CLASS=L with DISP=D attributes. These defaults can be modified by changing the options specified in the CEELOPT macro. This macro is generated when the installation default run-time options are assembled:
  - You use member CEEWCOPT.Z to assemble the installation default run-time options under CICS. For details, see “Changing the Installation-Wide Run-Time Options Default (CICS)” in the *LE/VSE Customization Guide*.
  - You use member CEEWDOPT.Z to assemble the installation default run-time options under batch. For details, see “Changing the

Installation-Wide Run-Time Options Default (BATCH)" in the *LE/VSE Customization Guide*.

- In the BATCH environment, the VSE/POWER LSTQ entry name is taken from the executing application job stream. The user-id is taken from any `FROM=` card specified in the active JOB JECL statement (when available). Otherwise, LE/VSE uses a default user-id of `LE$LSTQ$`. In the CICS environment, the VSE/POWER LSTQ entry name is built by using the failing CICS transaction identification name and the CICS terminal number that the user was signed-on to at the time of the failure. For example, if transaction `MENU` fails on terminal A001, the LSTQ entry name will be `MENUA001`.
- When the CICS Transaction Server is started, message DFHLD0107I may be issued for module CEELEDT if this module is not loaded in the SVA. This is an informational message only, and can be ignored. To stop the informational message from being issued, you can load module CEELEDT into the SVA and make the required CICS Transaction Server startup and CSD changes. The module will reside in SVA-31 storage.
- LSTQ support is not available in a pre-initialized environment.
- LSTQ is not supported for 4083 abends.

reg_stor_amount
Controls the amount of storage to be dumped around registers. The *reg_stor_amount* variable must be in the range 0-256 and indicates the storage in bytes to be dumped around each register. The *reg_stor_amount* value will be rounded up to the nearest multiple of 32. If you call the CEE5DMP service and do not require a dump storage around registers (regardless of the *reg_stor_amount* value), you must specify `REGSTor(0)` as a CEE5DMP option.

Usage Notes

- PL/I considerations— After a normal return from a PL/I ERROR ON-unit or from a PL/I FINISH ON-unit, LE/VSE 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.
- CICS considerations—The IBM-supplied default setting for TERMTHDACT under CICS is TERMTHDACT(TRACE,MSGFL,0).

  All TERMTHDACT output is written to the destination defined by the MSGFILE run-time option setting, except for the dump produced by the UADUMP option, which is a 4039 CICS transaction abend dump.

  TERMTHDACT(DUMP) is the setting you should use when attempting to determine the cause of errors in a CICS environment.

For More Information

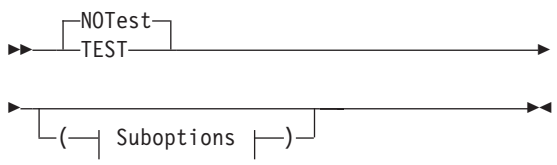
- For more information about the TRACE run-time option, see "TRACE" on page 48.
- For more information about the CEE5DMP service and its parameters, see "CEE5DMP—Generate Dump" on page 69.
- For more information about the TERMTHDACT run-time option and condition messages, see *LE/VSE Programming Guide*.
- For more information about the CESE transient data queue, see *LE/VSE Programming Guide*.

TEST | NOTEST

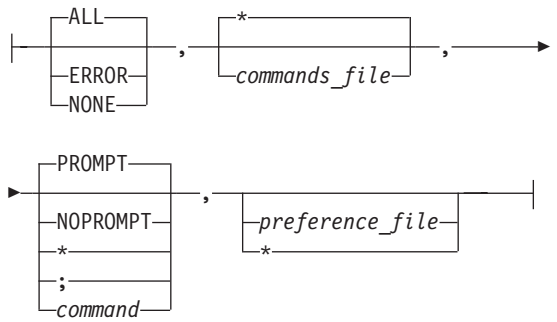
TEST specifies the conditions under which a debug tool (such as Debug Tool for VSE/ESA) assumes control when the user application is being initialized. Parameters of the TEST and NOTEST run-time options are merged as one set of parameters.

IBM-Supplied Default:
NOTEST(ALL,*,PROMPT,")

Syntax



Suboptions:



ALL
Specifies that either of the following will cause the debug tool to gain control even without a defined AT OCCURRENCE for a particular condition or AT TERMINATION:
- Any LE/VSE condition of severity 1 or above
- Application termination

ERROR
Specifies that only one of the following will cause the debug tool to gain control without a defined AT OCCURRENCE for a particular condition or AT TERMINATION:
- Any LE/VSE-defined error condition of severity 2 or higher
- Application termination

NONE
Specifies that no condition will cause the debug tool to gain control without a defined AT OCCURRENCE for a particular condition or AT TERMINATION.

***** (asterisk—in place of *commands_file*)
Specifies that no *commands_file* is supplied. The default device, as defined by your debug tool, is used as the source of the debug tool commands.

commands_file
A character string, up to 80 characters long, that you use to pass to the debug tool the filename of the primary commands file for this run. The syntax of the character string is defined by the debug tool you are using.

You can enclose *commands_file* in single or double quotes to distinguish it from the rest of the TEST | NOTEST suboption list.

PROMPT
Specifies that the debug tool is invoked at LE/VSE initialization.

NOPROMPT
Specifies that the debug tool is not invoked at LE/VSE initialization.

***** (asterisk—in place of PROMPT/NOPROMPT)
Specifies that the debug tool is not invoked at LE/VSE initialization; equivalent to NOPROMPT.

; (semicolon—in place of PROMPT/NOPROMPT)
Specifies that the debug tool is invoked at LE/VSE initialization; equivalent to PROMPT.

command
A character string, up to 250 characters long, that specifies a valid debug tool command. The command string can be enclosed in single or double quotes to distinguish it from the rest of the TEST parameter list; it cannot contain DBCS characters. Quotes are needed whenever the command list contains embedded blanks, commas, semicolons, or parentheses.

preference_file
A character string, up to 80 characters long, that you use to pass to the debug tool the filename of 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. The syntax of the character string is defined by the debug tool you are using.

You can enclose *preference_file* in single or double quotes to distinguish it from the rest of the TEST parameter list.

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

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 for VSE/ESA 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 run-time option setting.

Performance Consideration

To improve performance, use this option only while debugging.

For More Information

- For more information about the syntax of the TEST run-time option when using Debug Tool for VSE/ESA, see *Debug Tool for VSE/ESA User's Guide and Reference*
- For more information about creating and using a commands file and a preference file for Debug Tool for VSE/ESA, see *Debug Tool for VSE/ESA User's Guide and Reference*

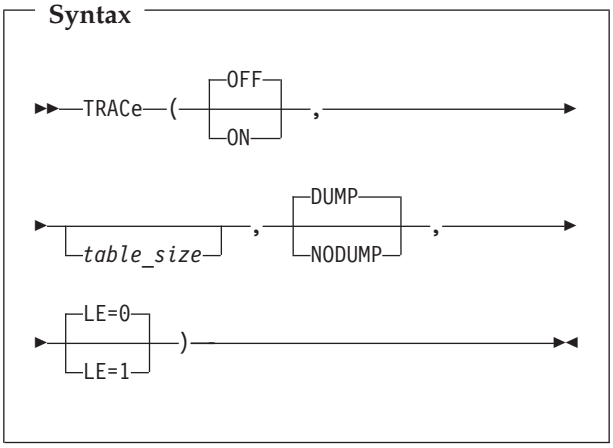
TRACE

TRACE controls run-time 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 LE/VSE 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 run-time option and whether TRACE is active and the DUMP suboption is specified.

IBM-Supplied Default:
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.

DUMP
Requests that an LE/VSE-formatted dump (containing the trace table) be taken at program termination regardless of the setting of the TERMTHDACT run-time option.

NODUMP
Requests that an LE/VSE-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 LE/VSE member libraries be recorded (such as, in the case of C, entry and exit of the `printf()` library function).

For More Information

- For more information about the dump contents, see "TERMTHDACT" on page 44.
- For more information about using the tracing facility, see *LE/VSE Debugging Guide and Run-Time Messages*.

TRAP

Trap specifies the level of condition handling that LE/VSE performs for user abends and program interrupts.

You must specify at least TRAP(ON,MIN) in order for applications to run successfully.

TRAP(ON,MAX) must be in effect for the ABTERMENC or ABPERC run-time options to have effect.

This option is similar to the STAE | NOSTAE run-time option previously offered by VS COBOL II, C/370, and DOS PL/I, and the SPIE | NOSPIE option offered by C/370:

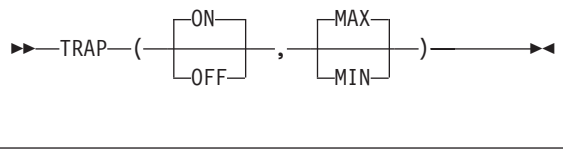
Table 18. TRAP Run-Time Option Settings

If...	then...
a single option is specified in input,	TRAP is set according to that option, TRAP(OFF) for NOSTAE or NOSPIE, TRAP(ON) for STAE or SPIE.
both options are specified in input,	TRAP is set ON, unless both options are negative, then TRAP is set OFF.
STAE is specified in one `#pragma runopts` statement, and NOSPIE in another,	the option in the last `#pragma runopts` determines the setting of TRAP.
multiple instances of STAE | NOSTAE are specified,	TRAP is set according to the last instance only. All others are ignored.
multiple instances of SPIE | NOSPIE are specified,	TRAP is set according to the last instance only. All others are ignored.
an options string has TRAP(ON) or TRAP(OFF) together with SPIE | NOSPIE and/or STAE | NOSTAE,	the TRAP setting takes preference over all others.

The use of the CEESGL callable service is not affected by this option.

IBM-Supplied Default: TRAP(ON,MAX)

Syntax



ON
Enables LE/VSE condition handling.

OFF
Disables LE/VSE condition handling. The MIN|MAX option is ignored when TRAP(OFF) is used.

MAX
Instructs LE/VSE to activate full condition handling. This will involve the use of both STXIT AB and STXIT PC processing. TRAP sub-option ON must also be specified for this option to have any effect.

MIN
Instructs LE/VSE not to use any STXIT AB processing for LE/VSE condition handling and to only use STXIT PC condition handling. This is required for internal LE/VSE failures that are part of application abend reporting and dump producing functions. TRAP sub-option ON must also be specified for this option to have any effect.

Note: Each of the HLLs will still issue STXIT AB's as required regardless of the TRAP run-time option setting.

Usage Notes

During normal processing, LE/VSE expects TRAP(ON,..) to be in effect for the application to run successfully. Use TRAP(ON,MIN) when a program exception needs to be analyzed. The use of TRAP(OFF,...) is not recommended and can cause unpredictable results to occur. Alternatively, an abnormal termination exit can be used to analyze the failure and report on the problem using information provided by LE/VSE to the exit.

Specifying TRAP(ON) without MIN or MAX is equivilent to specifying TRAP(ON,MAX).

The use of TRAP(ON,MIN) when an abnormal condition occurs within a user application will result in:

- Any user registered condition handlers will not be called. This includes handlers specified via the USRHDLR run-time option.
- LE DUMP processing will not be performed.
- ABTERMENC run-time option has no effect.
- ABPERC run-time option has no effect.
- No resources acquired by LE/VSE are released.
- Files opened by HLLs are not closed and this may result in the loss of data.
- The abnormal termination exit is not driven.
- The assembler user exit is not called for enclave termination.
- Debug Tool is not notified of the error.
- No storage or run-time options reports are generated.
- No LE/VSE messages are produced.
- For a user program interrupt, or an unexpected LE/VSE program interrupt, LE/VSE will try to re-execute the failing instruction in an attempt

to invoke normal VSE abnormal termination semantics. If this is not possible, a JDUMP macro will be issued to terminate the enclave and to produce a system dump.

Note: LE/VSE internal condition handling is enabled but any user program interrupts or abends are not handled by LE/VSE condition management.

Running with TRAP(OFF) can cause many side effects because LE/VSE requires internal condition handling to successfully execute. If you run with TRAP(OFF), you can get failures even if you do not encounter a software-raised condition, program check or abend. If you do encounter a program check or abend with TRAP(OFF) in effect, the following will occur:

- The ABTERMENC run-time option will have no effect.
- The ABPERC run-time option will have no effect.
- Resources acquired by LE/VSE are not freed.
- Files opened by HLLs are not closed so data may 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 Debug Tool is not notified of the error.
- No storage report or run-time options report is generated.

The enclave terminates abnormally if such conditions are raised.

TRAP(ON,MAX) must be in effect when you want to use the CEEBXITA assembler user exit for enclave initialization to specify a list of VSE cancel codes, program interruption code and user abend codes that LE/VSE exempts from normal condition handling.
When TRAP(ON,MAX) is in effect and an abnormal condition occurs, if the VSE cancel code, program-interruption code, or user abend code is in the CEEAUE_CODES list in CEEBXITA, LE/VSE exempts the condition from normal condition handling. Normal LE/VSE condition handling is never invoked to handle these conditions. This feature is useful when you do not want LE/VSE condition handling to intervene for certain abnormal conditions, or when you want to prevent invocation of the abnormal termination exit for such conditions.
When TRAP(OFF) or TRAP(ON,MIN) are set and there is a program interrupt, or an abend, the assembler user exit is not driven at termination. Any information provided in CEEAUE_CODES is ignored.

CICS Considerations

The MIN|MAX option is ignored under CICS. If you specify TRAP(OFF) in a CICS environment, LE/VSE does not produce any messages or dumps for conditions raised by program interrupts or transaction abends. The standard CICS system action occurs. However abends not caused by the application program, can occur, as internal LE/VSE condition handling has been disabled by the use of TRAP(OFF). For applications to run successfully under CICS, TRAP(ON,MAX) is required.

Performance Considerations

When using COBOL internal sorts, with a SORT product's STXIT option activated either by default or by design, performance benefits can be achieved by using TRAP(ON,MIN) for the application. Alternatively, if possible, set the SORT product's option to NOSTXIT (or to MINSTXIT in the case of DFSORT/VSE) which will allow the use of TRAP(ON,MAX) without impacting performance.

For More Information

For more information about the:

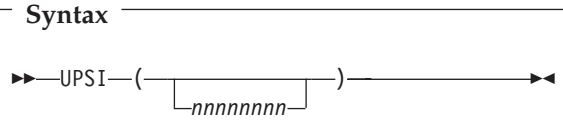
- ABPERC and ABTERMENC run-time options, see "ABPERC" on page 19 and "ABTERMENC" on page 20.
- CEESGL callable service, see "CEESGL—Signal a Condition" on page 193.
- CEEBXITA assembler user exit, refer to *LE/VSE Programming Guide*.

UPSI (COBOL Only)

UPSI sets the eight UPSI switches on or off for applications that use COBOL routines.

IBM-Supplied Default: UPSI(00000000)

Syntax



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).

The IBM-supplied default setting is UPSI(00000000).

Usage Notes

- Do not confuse the LE/VSE UPSI run-time option with the job control UPSI statement. The UPSI switches set by the job control UPSI statement are not available to COBOL routines under LE/VSE.
- When you specify this option in CEEDOPT or CEEUOPT, use the CEEXOPT macro syntax to specify UPSI with a string of eight binary-valued flags; for example, UPSI=(00000000). Use UPSI, not followed by a string, only on the command line.

For More Information

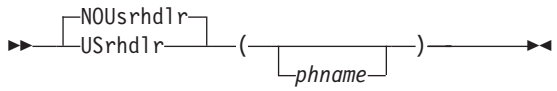
- For more information on how COBOL routines access the UPSI switches, see *IBM COBOL for VSE/ESA Programming Guide*
- For more information about CEEDOPT, see *LE/VSE Customization Guide*.
- For more information about CEEUOPT, see *LE/VSE Programming Guide*.

USRHDLR | NOUSRHDLR

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.

IBM-Supplied Default: NOUSRHDLR

Syntax



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.

phname
The name of a phase that contains the user condition handler that is to be registered at stack frame 0.

Usage Notes

- User Handler module names of YES, NO, ON and OFF, are no longer permitted
- The user condition handler specified by the USRHDLR run-time option must be in a separate phase rather than be linkedited with the rest of the application.
- The user condition handler *phname* is invoked for conditions that are still unhandled after being presented to condition handlers for the main program.
- Restriction - if USRHDLR is in effect, you cannot resume execution in the program in which the condition occurs. This includes calls in the condition handler to CEEMRCR and CEEMRCE.
- You can use a user condition handler registered with the USRHDLR run-time 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 with the USRHDLR run-time option, is not presented to any other user condition handler.
- The loading of the user condition handler *phname* occurs only when that user condition handler needs to be invoked the first time.
- PL/I Consideration - Although PL/I cannot use the CEEHDLR callable service to register a user-written condition handler, PL/I can use the USRHDLR run-time option.
- CICS Consideration - *phname* must be defined in the CICS System Definition File (CSD).

For More Information

For more information on registering a user condition handler, see:
- "CEEHDLR—Register User-Written Condition Handler" on page 146.
- *LE/VSE Programming Guide*.

XUFLOW

XUFLOW specifies whether an exponent underflow causes a program interrupt. An exponent underflow occurs when a floating point number becomes too small to be represented.

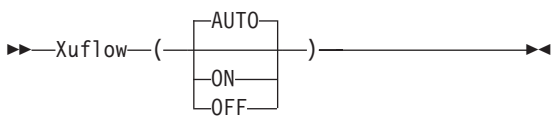
The underflow setting is determined at enclave initialization and is updated when new languages are introduced into the application (via fetch or dynamic call, for example). Otherwise, it does not vary while the application is running.

LE/VSE preserves the language semantics for C and COBOL regardless of the XUFLOW setting. LE/VSE preserves the language semantics for PL/I only when XUFLOW is set to AUTO or ON. LE/VSE does not preserve the language semantics for PL/I when XUFLOW is set to OFF.

An exponent underflow caused by a C or COBOL routine does not cause a condition to be raised.

IBM-Supplied Default: XUFLOW(AUTO)

Syntax



AUTO
An exponent underflow causes or does not cause a program interrupt dynamically, based upon the HLLs that make up the application. Enablement is determined without user intervention.

XUFLOW(AUTO) causes condition management to process underflows only in those applications where the semantics of the application languages require it. Normally, XUFLOW(AUTO) provides the best efficiency while meeting language semantics.

ON
An exponent underflow causes a program interrupt.

XUFLOW(ON) causes condition management to process underflows regardless of the mix of languages; therefore, this setting might be less efficient in applications that consist of languages not requiring underflows to be processed by condition management.

OFF
An exponent underflow does not cause a program interrupt; the hardware takes care of the underflow.

When you set XUFLOW to OFF, the hardware processes exponent underflows. This is more efficient than condition handling to process the underflow.

Usage Notes

- PL/I consideration—When setting XUFLOW to OFF, be aware that the semantics of PL/I require the underflow to be signaled.

Language Run-Time Option Mapping

Under LE/VSE, there is one set of run-time options. LE/VSE parses and merges the run-time options. These options are processed at the enclave level and allow you to control many aspects of the LE/VSE environment.

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

Tables of pre-LE/VSE, HLL-specific options and their LE/VSE equivalents are provided below. The mapping is performed automatically by LE/VSE except where noted.

Table 19. C/370 and LE/VSE Options

C/370 Option	LE/VSE Equivalent	Notes
ISAINC (*incr_size*)	STACK (*incr_size*)	The C/370 ISAINC run-time option is mapped to the LE/VSE STACK run-time option for compatibility. It affects all languages in the enclave.
ISASIZE (*init_size*)	STACK (*init_size*)	The C/370 ISASIZE run-time option is mapped to the LE/VSE STACK run-time option for compatibility. It affects all languages in the enclave.
LANGUAGE	NATLANG	The C/370 LANGUAGE run-time option is mapped to the LE/VSE NATLANG run-time option for compatibility. It affects all languages in the enclave.
REPORT	RPTSTG(ON)	The C/370 REPORT run-time option is mapped to the LE/VSE RPTSTG(ON) run-time option for compatibility. It affects all languages in the enclave.
NOREPORT	RPTSTG(OFF)	The C/370 NOREPORT run-time option is mapped to the LE/VSE RPTSTG(OFF) run-time option for compatibility. It affects all languages in the enclave.
SPIE	TRAP(ON,MAX)	The C/370 SPIE run-time option is mapped to the LE/VSE TRAP(ON,MAX) run-time option for compatibility. It affects all languages in the enclave. The mapping of SPIE might differ depending upon other options specified. For more information, see "TRAP" on page 48.
NOSPIE	TRAP(OFF)	The C/370 NOSPIE run-time option is mapped to the LE/VSE TRAP(OFF) run-time option for compatibility. It affects all languages in the enclave. The mapping of NOSPIE might differ depending upon other options specified. For more information, see "TRAP" on page 48.
STAE	TRAP(ON,MAX)	The C/370 STAE run-time option is mapped to the LE/VSE TRAP(ON,MAX) run-time option for compatibility. It affects all languages in the enclave. The mapping of STAE might differ depending upon other options specified. For more information, see "TRAP" on page 48.
NOSTAE	TRAP(ON,MIN)	The C/370 NOSTAE run-time option is mapped to the LE/VSE TRAP(ON,MIN) run-time option for compatibility. It affects all languages in the enclave. The mapping of NOSTAE might differ depending upon other options specified. For more information, see "TRAP" on page 48.

Table 20. DOS/VS COBOL and LE/VSE Options

DOS/VS COBOL Option	LE/VSE Equivalent	Notes
A (SYSPARM)	AIXBLD	The LE/VSE AIXBLD run-time option is compatible with the DOS/VS COBOL SYSPARM='A' run-time option. It affects only COBOL programs in the enclave.

Table 20. DOS/VS COBOL and LE/VSE Options (continued)

DOS/VS COBOL Option	LE/VSE Equivalent	Notes
NA (SYSPARM)	NOAIXBLD	The LE/VSE NOAIXBLD run-time option is compatible with the DOS/VS COBOL SYSPARM='NA' run-time option. It affects only COBOL programs in the enclave.
D (SYSPARM)	DEBUG	The LE/VSE DEBUG run-time option is compatible with the DOS/VS COBOL SYSPARM='D' run-time option. It affects only COBOL programs in the enclave.
ND (SYSPARM)	NODEBUG	The LE/VSE NODEBUG run-time option is compatible with the DOS/VS COBOL SYSPARM='ND' run-time option. It affects only COBOL programs in the enclave.
UPSI	UPSI	The LE/VSE UPSI run-time option replaces the DOS/VS COBOL UPSI run-time option provided by the `// UPSI` job control statement. The UPSI switches set by the `// UPSI` job control statement are not available to COBOL programs under LE/VSE. If UPSI job control switches are required, you can refer to an example LE assembler user exit provided on the LE/VSE web-site. This example shows how to provide JCL UPSI switches to COBOL/VSE programs.

Table 21. VS COBOL II and LE/VSE Options

VS COBOL II Option	LE/VSE Equivalent	Notes
AIXBLD	AIXBLD	The LE/VSE AIXBLD run-time option is compatible with the VS COBOL II AIXBLD run-time option. It affects only COBOL programs in the enclave.
NOAIXBLD	NOAIXBLD	The LE/VSE NOAIXBLD run-time option is compatible with the VS COBOL II NOAIXBLD run-time option. It affects only COBOL programs in the enclave.
DEBUG	DEBUG	The LE/VSE DEBUG run-time option is compatible with the VS COBOL II DEBUG run-time option. It affects only COBOL programs in the enclave.
NODEBUG	NODEBUG	The LE/VSE NODEBUG run-time option is compatible with the VS COBOL II NODEBUG run-time option. It affects only COBOL programs in the enclave.
LANGUAGE	NATLANG	The VS COBOL II LANGUAGE run-time option is mapped to the LE/VSE NATLANG run-time option for compatibility. It affects all languages in the enclave.
LIBKEEP	Not applicable	There is no LE/VSE equivalent for the VS COBOL II LIBKEEP run-time option. To obtain similar performance function, use the Library Routine Retention (LRR) feature described in the *LE/VSE Programming Guide*.
NOLIBKEEP	Not applicable	There is no LE/VSE equivalent for the VS COBOL II NOLIBKEEP run-time option.
MIXRES	Not applicable	There is no LE/VSE equivalent for the VS COBOL II MIXRES run-time option. MIXRES applications supported by LE/VSE always exhibit RES behavior.
NOMIXRES	Not applicable	There is no LE/VSE equivalent for the VS COBOL II NOMIXRES run-time option. MIXRES applications supported by LE/VSE always exhibit RES behavior.
RTEREUS	RTEREUS	The LE/VSE RTEREUS run-time option is compatible with the VS COBOL II RTEREUS run-time option. The RTEREUS option is intended for use when the main program of an enclave is a COBOL program. The RTEREUS option can cause problems for HLLs other than COBOL.
NORTEREUS	NORTEREUS	The VS COBOL II NORTEREUS run-time option is compatible with the VS COBOL II NORTEREUS run-time option.
SIMVRD	Not applicable	There is no LE/VSE equivalent for the VS COBOL II SIMVRD run-time option.

Table 21. VS COBOL II and LE/VSE Options (continued)

VS COBOL II Option	LE/VSE Equivalent	Notes
NOSIMVRD	Not applicable	There is no LE/VSE equivalent for the VS COBOL II NOSIMVRD run-time option.
SPOUT	RPTOPTS(ON) RPTSTG(ON)	The VS COBOL II SPOUT run-time option is mapped to the LE/VSE RPTOPTS(ON) and RPTSTG(ON) run-time options for compatibility. It affects all languages in the enclave.
NOSPOUT	RPTOPTS(OFF) RPTSTG(OFF)	The VS COBOL II NOSPOUT run-time option is mapped to the LE/VSE RPTOPTS(OFF) and RPTSTG(OFF) run-time options for compatibility. It affects all languages in the enclave.
SSRANGE	CHECK(ON)	The VS COBOL II SSRANGE run-time option is mapped to the LE/VSE CHECK(ON) run-time option for compatibility. It affects only COBOL programs in the enclave.
NOSSRANGE	CHECK(OFF)	The VS COBOL II NOSSRANGE run-time option is mapped to the LE/VSE CHECK(OFF) run-time option for compatibility. It affects only COBOL programs in the enclave.
STAE	TRAP(ON,MAX)	The VS COBOL II STAE run-time option is mapped to the LE/VSE TRAP(ON,MAX) run-time option for compatibility. It affects all languages in the enclave. The mapping of STAE might differ depending upon other options specified. For more information, see "TRAP" on page 48.
NOSTAE	TRAP(ON,MIN)	The VS COBOL II NOSTAE run-time option is mapped to the LE/VSE TRAP(ON,MIN) run-time option for compatibility. It affects all languages in the enclave. The mapping of NOSTAE might differ depending upon other options specified. For more information, see "TRAP" on page 48.
UPSI	UPSI	The VS COBOL II UPSI option is processed for compatibility.
WSCLEAR	STORAGE(00)	The VS COBOL II WSCLEAR run-time option is not supported under LE/VSE. For behavior similar to that produced by the VS COBOL II WSCLEAR run-time option, use the LE/VSE STORAGE(00) run-time option.
NOWSCLEAR	STORAGE(NONE)	The VS COBOL II NOWSCLEAR run-time option is not supported under LE/VSE. For behavior similar to that produced by the VS COBOL II NOWSCLEAR run-time option, use the LE/VSE STORAGE(NONE) run-time option.

Table 22. DOS PL/I and LE/VSE Options

DOS PL/I Option	LE/VSE Equivalent	Notes
COUNT	Not applicable	There is no LE/VSE equivalent for the DOS PL/I COUNT run-time option.
NOCOUNT	Not applicable	There is no LE/VSE equivalent for the DOS PL/I NOCOUNT run-time option.
FLOW	Not applicable	There is no LE/VSE equivalent for the DOS PL/I FLOW run-time option.
NOFLOW	Not applicable	There is no LE/VSE equivalent for the DOS PL/I NOFLOW run-time option.
ISASIZE (*init_size*)	STACK (*init_size*)	The DOS PL/I ISASIZE run-time option is mapped to the LE/VSE STACK run-time option for compatibility. It affects all languages in the enclave.
REPORT	RPTSTG(ON)	The DOS PL/I REPORT run-time option is mapped to the LE/VSE RPTSTG(ON) run-time option for compatibility. It affects all languages in the enclave.
NOREPORT	RPTSTG(OFF)	The DOS PL/I NOREPORT run-time option is mapped to the LE/VSE RPTSTG(OFF) run-time option for compatibility. It affects all languages in the enclave.

Table 22. DOS PL/I and LE/VSE Options (continued)

DOS PL/I Option	LE/VSE Equivalent	Notes
STAE	TRAP(ON,MAX)	The DOS PL/I STAE run-time option is mapped to the LE/VSE TRAP(ON,MAX) run-time option for compatibility. It affects all languages in the enclave. The mapping of STAE might differ depending upon other options specified. For more information, see "TRAP" on page 48.
NOSTAE	TRAP(ON,MIN)	The DOS PL/I NOSTAE run-time option is mapped to the LE/VSE TRAP(ON,MIN) run-time option for compatibility. It affects all languages in the enclave. The mapping of NOSTAE might differ depending upon other options specified. For more information, see "TRAP" on page 48.

Chapter 3. LE/VSE Callable Services

LE/VSE callable services provide functions that your pre-LE/VSE run-time libraries might not provide. You can use these services alone or with LE/VSE run-time options, which customize your run-time environment.

This chapter provides syntax and examples of LE/VSE callable services. You can invoke LE/VSE callable services from applications generated with any of the LE/VSE-conforming language compilers listed in Table 1 on page xvi, or assembler routines coded using the CEEENTRY and associated macros and assembled using High Level Assembler.

You can also use dynamic calls from VS COBOL II programs to call the following LE/VSE date and time callable services: CEE5CTY, CEECBLDY, CEEDATE, CEEDATM, CEEDAYS, CEEDYWK, CEEGMT, CEEGMTO, CEEISEC, CEELOCT, CEEQCEN, CEESCEN, CEESECI, and CEESECS.

Note: You cannot use static calls from VS COBOL II programs to call **any** LE/VSE callable services, and you should not use dynamic calls from VS COBOL II programs to call any LE/VSE callable services other than those listed above.

For guidelines about writing your own callable services, see *LE/VSE Programming Guide*.

For a list of LE/VSE callable services, see Chapter 1, "LE/VSE Quick Reference Tables," on page 1.

General Usage Notes for Callable Services

- You can invoke callable services from any LE/VSE-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 have names beginning with CEE are intended to be consistent across all platforms that Language Environment supports. Callable services that have names beginning with CEE5 are specific to the VSE platform supported by LE/VSE.
- Routines that invoke callable services do not need to be AMODE(31). AMODE switching is performed implicitly without any action required by the calling routine, if you specify the ALL31(OFF) run-time option (see "ALL31" on page 22).

  However, if AMODE switching occurs and your program makes many calls to LE/VSE callable services, the switching time can slow down your application. Run with AMODE(31), if possible, to avoid unnecessary mode switching.
- Under LE/VSE, all parms are passed by reference, indirectly.

Invoking Callable Services

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

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

Header, Copy, or Include Files

Many of the programming examples in this chapter 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 LE/VSE callable services that you would otherwise need to code in your program. They can also help you reduce errors by verifying correct usage of LE/VSE callable services at compile time.

The names and descriptions of the files imbedded in the callable service examples in this chapter are provided in Table 23 on page 58.

Table 23. Files Used in C, COBOL, and PL/I Examples

File Name	Description
ceeedcct.h	C declarations for LE/VSE symbolic feedback codes
leawi.h	Declarations of LE/VSE callable services and OMIT_FC, which is used to explicitly omit the *fc* parm, for routines written in C
CEEIGZCI	COBOL declarations for condition information block
CEEIGZCT	COBOL declarations for LE/VSE symbolic feedback codes
CEEIGZLC	COBOL declarations for locale category constants LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.
CEEIGZTD	COBOL declaration for TD_STRUCT structure needed for CEEFTDS calls
CEEIGZNM	COBOL declaration for NM_STRUCT structure needed for CEELCNV calls
CEEIGZDT	COBOL declaration for DTCONV structure needed for CEEQDTC calls
CEEIBMAW	LE/VSE callable service declarations and data attribute declarations for routines written in PL/I
CEEIBMCI	PL/I declarations for condition information block
CEEIBMCT	PL/I declarations for LE/VSE symbolic feedback codes
CEEIBMLC	PL/I declarations for locale category constants LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.
CEEIBMTD	PL/I declaration for TD_STRUCT structure needed for CEEFTDS calls
CEEIBMNM	PL/I declaration for NM_STRUCT structure needed for CEELCNV calls
CEEIBMDT	PL/I declaration for DTCONV structure needed for CEEQDTC calls

Notes:
- A symbolic feedback code is a symbolic representation of a condition token.
- PL/I routines that imbed CEEIBMAW require the MACRO compiler option.
- COBOL does not require declarations of external programs; therefore, you do not need declarations of LE/VSE callable services for programs written in COBOL.

All these files are provided with LE/VSE. If LE/VSE is installed in the default installation sublibraries, they are contained in the PRD2.SCEEBASE sublibrary.

You can imbed these files using the statement appropriate for the language your routine is written in, as shown in Table 24:

Table 24. Imbedding Files in Your Routines

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

Examples of these statements are shown in the following sections on syntax.

Sample Programs

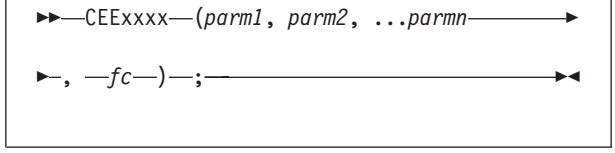
Most of the callable service descriptions in this chapter include sample routines written in C, COBOL, and PL/I, as appropriate.

> The sample routines in this chapter are provided in source format with the LE/VSE product. If LE/VSE is installed in the default installation sublibraries, they are located in the PRD2.SCEEBASE sublibrary.

C Syntax

In C, use the following syntax to invoke an LE/VSE callable service with a feedback code in effect:

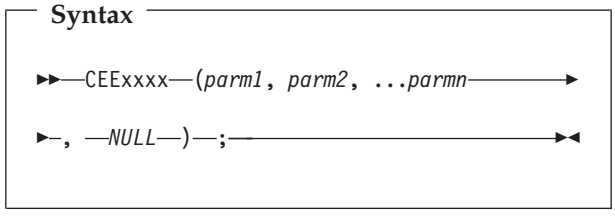
Syntax



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



See "Parameter List for Invoking Callable Services" on page 60 for a description of this syntax.

LE/VSE 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.

You can use the header file `leawi.h` to declare LE/VSE callable services, and the header file `ceeedcct.h` to declare LE/VSE symbolic feedback codes, as shown below:

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

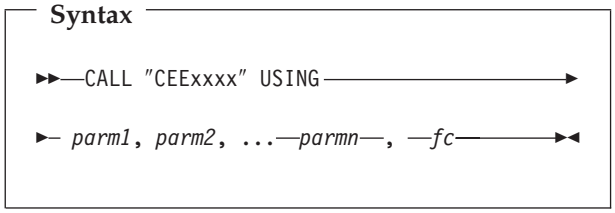
Figure 4. Sample Callable Services Invocation Syntax for C

To help in checking the success of your applications, LE/VSE provides the `_FBCHECK()` function 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. See the sample C routines in this chapter for examples of how to use `_FBCHECK()`.

COBOL Syntax

In COBOL, use the following syntax to invoke LE/VSE callable services:

Syntax



You can call LE/VSE services either statically or dynamically from COBOL applications.

See “Parameter List for Invoking Callable Services” on page 60 for a description of this syntax.

You can use the copy file CEEIGZCT to declare symbolic LE/VSE feedback codes, in conjunction with a COBOL call to an LE/VSE callable service to return a feedback code, as shown below:

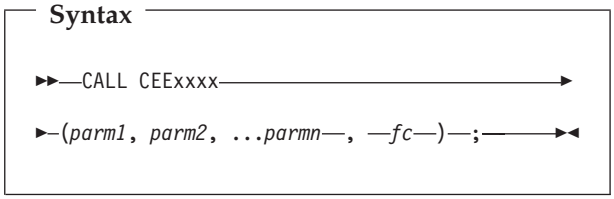
```
COPY CEEIGZCT
CALL "CEExxxx" USING parm1, parm2, ... parmn, fc
```

Figure 5. Sample Callable Services Invocation Syntax for COBOL

PL/I Syntax

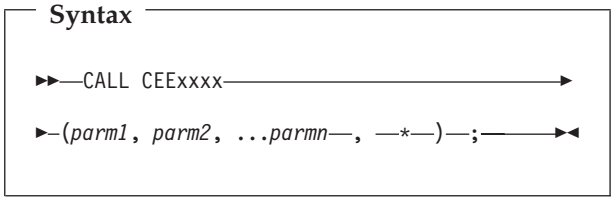
In PL/I, use the following syntax to invoke an LE/VSE callable service with a feedback code in effect:

Syntax



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

Syntax



Note that you cannot invoke callable services as function references.

See “Parameter List for Invoking Callable Services” on page 60 for a description of this syntax.

The value of register 15 upon return from any LE/VSE 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 LE/VSE callable services, be sure to specify all required arguments in the CALL statement. Figure 6 on page 60 illustrates a call in which the feedback code to CEEDATE has been incorrectly omitted.

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

Figure 6. An Invalid Call that Omits the fc Parameter

Figure 7 illustrates valid calls:

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

Figure 7. Valid Calls that Use the Optional fc Parameter

You can use the include file CEEIBMAW to declare LE/VSE callable services, and the include file CEEIBMCT to declare LE/VSE symbolic feedback codes, as shown below:

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

Figure 8. Sample Callable Services Invocation Syntax for PL/I

To help in checking the success of your applications, LE/VSE provides the FBCHECK function in the CEEIBMCT include file. FBCHECK compares a feedback code against a condition token you supply to determine whether you receive the feedback code you should. See the sample PL/I routines in this chapter for examples of how to use FBCHECK.

Parameter List for Invoking Callable Services

This section describes the syntax and parameters you need to invoke LE/VSE 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 57 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 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 the parm is omitted.

fc A feedback code that indicates the result of the service. *fc* can be omitted when you use C and PL/I. COBOL does not allow omitted parameters.

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 LE/VSE 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 LE/VSE 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 25 includes data type definitions and their descriptions for COBOL, C, and PL/I:

Table 25. Data Type Definitions across LE/VSE-Conforming HLLs

Data Type	Description	COBOL	C	PL/I
INT2	A 2-byte signed integer	PIC S9(4) USAGE IS BINARY	signed short	REAL FIXED BINARY (15,0)
INT4	A 4-byte signed integer	PIC S9(9) USAGE IS BINARY	signed int	REAL FIXED BINARY (31,0)

Table 25. Data Type Definitions across LE/VSE-Conforming HLLs (continued)

Data Type	Description	COBOL	C	PL/I
FLOAT4	A 4-byte single-precision floating-point number	COMP-1	float	REAL FLOAT BINARY (21) or REAL FLOAT DECIMAL (6)
FLOAT8	An 8-byte double-precision floating-point number	COMP-2	double	REAL FLOAT BINARY (53) or REAL FLOAT DECIMAL (16)
FLOAT16	A 16-byte extended-precision floating-point number	Not available	long double	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.	Not available	Not available	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.	Not available	Not available	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.	Not available	Not available	COMPLEX FLOAT DECIMAL (33)
POINTER	A platform-dependent address pointer	USAGE IS POINTER	void *	POINTER
CHAR*n*	A string (character array) of length *n*	PIC X(n)	char[n]	CHAR(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).`
			`struct _VSTRING(`
			`_INT2 length;`
			`char string[1];`
			`}string-in;`
			`char string-out[80];`
				`DCL string_in`
				`CHAR(n) VARYING;`
				`DCL string_out CHAR(80);`

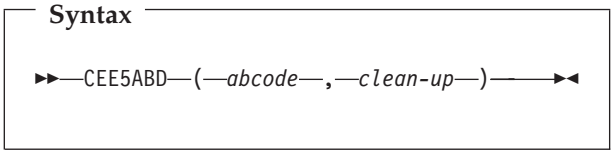
FEED_BACK	A mapping of the condition token (fc)	`Case 1:`
		`01 FC`
		`02 SEV PIC S9(4)`
		`USAGE IS BINARY.`
		`02 MSGNO PIC S9(4)`
		`USAGE IS BINARY.`
		`02 FLGS PIC X(1).`
		`02 FACID PIC X(3).`
		`02 ISI PIC X(4).`
		
		`Case 2:`
		`01 FC`
		`02 CLASS-CODE PIC X(2).`
		`02 CAUSE-CODE PIC X(2).`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

CEE5ABD—Terminate Enclave with an Abend

CEE5ABD requests that LE/VSE 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



abcode
A fullword integer, no greater than 4095, specifying the abend code that is issued. When executing in the batch environment, this fullword integer is converted to the equivalent EBCDIC, and included in abnormal termination message CEE3322C. When executing under CICS, this fullword integer is converted to the equivalent EBCDIC and the abend is then issued.

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

Value	Meaning
0	Issue the abend without clean-up
1	Issue the abend with normal enclave termination processing.

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

If *clean-up* is 0, no LE/VSE dump is generated. A system dump, however, is requested when issuing the abend. Under CICS, a transaction dump is taken. To obtain a dump in the batch environment, specify the VSE DUMP option in your JCL.

If *clean-up* is 0, LE/VSE condition handling is disabled for the current enclave and termination activities are not performed. Event handlers are not driven; 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-LE/VSE abend. Its processing is affected by the ABPERC and TRAP options, the user abend codes specified 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 an LE/VSE dump is produced, depending on the setting of the TERMTHDACT option; event handlers are driven; and user exits are invoked. Assembler user exit settings control whether the application actually terminates with an abend.

Usage Notes

These usage notes only apply to CICS:

- when *clean-up* is set to 0, LE will terminate the enclave immediately via an EXEC CICS ABEND ABCODE(xxxx) CANCEL. This means that any active EXEC CICS HANDLE ABEND commands in the application will not receive control.
- if *clean-up* is set to 1, LE will abend the enclave using an EXEC CICS ABEND ABCODE(xxxx), omitting the CANCEL option. This will allow any active EXEC CICS HANDLE ABEND commands will receive control.
- to ensure that an abend and handle-abend loop do not occur, it is the application programmers responsibility to ensure that programs or procedures that receive control from a call to CEE5ABD with *clean-up* set to 1, terminate the transaction via either:
  - EXEC CICS ABEND ABCODE(xxxx) CANCEL.
  - by calling CEE5ABD with CLEAN-UP set to 0.

  You might also refer to the information covering "HANDLE ABEND" in your *CICS Application Programming Guide* documentation.

For More Information

- For more information about the ABPERC run-time option, see "ABPERC" on page 19.
- For more information about the TRAP run-time option, see "TRAP" on page 48.
- For more information about the TERMTHDACT run-time option, see "TERMTHDACT" on page 44.

Examples

- **C Example**

```
/*Module/File Name: EDC5ABD   */

#include <leawi.h>

int main(void) {

  _INT4 code, timing;

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

CEE5ABD(&code,&timing);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5ABD
      ********************************************
      ** CBL5ABD - Call CEE5ABD to terminate the  **
      **           enclave with an abend          **
      ********************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5ABD.


       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 'CEE5ABD' USING ABDCODE , TIMING.

                GOBACK.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5ABD                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEE5ABD - terminate enclave with an    */
 /**                     abend                        */
 /**                                                  */
 /** In this example, CEE5ABD is called with a        */
 /** timing value of 1.  This requests an abend that */
 /** is deferred until clean-up takes place.          */
 /**                                                  */
 /*****************************************************/
 PLI5ABD: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL ABDCODE INT4;
    DCL TIMING  INT4;

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

    /* Call CEE5ABD to request an abend 3333 with    */
    /*    clean-up                                   */
    CALL CEE5ABD ( ABDCODE, TIMING );

 END PLI5ABD;
```

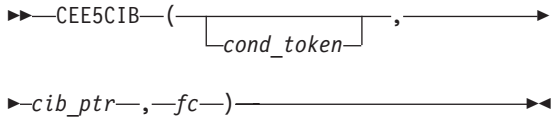
CEE5CIB—Return Pointer to Condition Information Block

CEE5CIB returns a pointer to a condition information block (CIB) associated with a given condition token. Use this service only during condition handling.

For COBOL applications, the COPY file CEEIGZCI (in the PRD2.SCEEBASE sublibrary) maps the CIB. For PL/I applications, the include file CEEIBMCI (in the PRD2.SCEEBASE sublibrary) maps the CIB. For C applications, the header file `leawi.h` (in the PRD2.SCEEBASE sublibrary) defines the structure `_CEECIB` that maps the CIB.

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



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

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

fc
A 12-byte feedback code, optional in some languages, that indicates the results of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

Symbolic Feedback Code	Severity	Message Number	Message Text
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 Note

- After the condition handling functions return control to your application, *cib_ptr* is no longer valid

For More Information

- For more information about the CEE5CIB callable service, see *LE/VSE Programming Guide*.

Examples

- **C Example**

```
/*Module/File Name: EDC5CIB   */

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);

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;

   CEE5CIB(fc, &cib_ptr, &cibfc);

  /* verify that CEE5CIB was successful */
  if ( _FBCHECK ( cibfc , CEE000 ) != 0 ) {
  printf("CEE5CIB 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;
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5CIB
      ************************************************
      **                                            **
      ** CBL5CIB - Register a condition handler     **
      **           that will call CEE5CIB to get    **
      **           a Condition Information Block     **
      **           (CIB) for the condition.         **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5CIB.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ROUTINE  PROCEDURE-POINTER.
       01  TOKEN                   PIC S9(9) BINARY.
       01  FC                      PIC X(12).
       PROCEDURE DIVISION.
       PARA-CBL5CIB.
           SET ROUTINE TO ENTRY 'HANDLER'.
           CALL 'CEEHDLR' USING ROUTINE, TOKEN, FC.

           GOBACK.
       END PROGRAM CBL5CIB.

     CBL LIB,APOST
         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 'CEE5CIB' 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.
```

- **PL/I Example**

```
*PROCESS OPT(0), MACRO;
 /* Module/File Name: IBM5CIB                        */
 /*************************************************/
 /**                                              **/
 /** Function: CEE5CIB - example of CEE5CIB       **/
 /**                     invoked from PL/I ON-unit  **/
 /**                                              **/
 /*************************************************/

 IBM5CIB:  PROCEDURE  OPTIONS(MAIN);

   %INCLUDE CEEIBMAW;
```

```
%INCLUDE CEEIBMCT;
%INCLUDE CEEIBMCI;

DECLARE
   CIB_PTR     POINTER,
   01 FC       FEEDBACK,
   divisor     FIXED BINARY(31) INITIAL(0);

ON  ZERODIVIDE  BEGIN;

   CALL CEE5CIB(*, 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( 'CEE5CIB failed with msg '
         || FC.MsgNo );
      END;

   END /* ON ZeroDivide */;


divisor = 15 / divisor  /* signals ZERODIVIDE */;

END IBM5CIB;
```

CEE5CTY—Set Default Country

CEE5CTY 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.

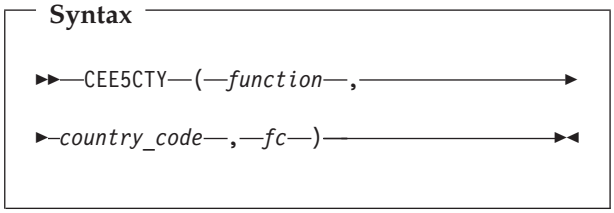
The current national country setting also affects the format of the date and time in the reports generated by the RPTOPTS and RPTSTG run-time options.

CEE5CTY also affects the default symbols for the NLS and date and time callable services.

CEE5CTY affects only the LE/VSE NLS and date and time services, not the LE/VSE locale callable services or C locale-sensitive functions.

If you need to override your default country setting, you can do so with CEE5CTY or the run-time option COUNTRY. For example, the code for the United States is specified as the default at installation. If in a certain application you want to use the French defaults, however, you could use CEE5CTY to SET France as the national country setting. Then when you again wanted the defaults for the United States, you would PUSH the code for United States back to the top of the stack.

Syntax



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. It is used in the following ways for the different *functions:*

If *function* is specified as:	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 227 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 CEE5CTY. The current country code was returned in the country code parameter.
CEE3C0	3	3456	The country code *country-code* for the PUSH or SET function for CEE5CTY was invalid. No operation was performed.
CEE3C1	3	3457	The function *function* specified for CEE5CTY was not recognized. No operation was performed.

Usage Notes

- The bytes X'0E' and X'0F' representing shift-out and shift-in codes are not affected by any *country_code* setting.
- C consideration—LE/VSE provides locales used in 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 CEE5CTY callable service. CEE5CTY affects only LE/VSE NLS and date and time services. `setlocale()` and CEESETL affect only C locale-sensitive functions and LE/VSE locale callable services.

  To ensure that all settings are correct for your country, use CEE5CTY and either CEESETL or `setlocale()`.

For More Information

- For more information about the RPTOPTS and RPTSTG run-time options, see "RPTOPTS" on page 37 and "RPTSTG" on page 38.
- For a list of the default settings for a specified country, see Table 32 on page 227.
- For more information about the COUNTRY run-time option, refer to "COUNTRY" on page 26.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Programming Guide*.

Examples

- **C Example**

```
/*Module/File Name: EDC5CTY   */

#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 */
  CEE5CTY(&function,country,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE5CTY 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 */
     CEE5CTY(&function,country,&fc);

     if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEE5CTY failed with message number %d\n",
                fc.tok_msgno);
        exit(2999);
     }

  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5CTY
      ************************************************
      **                                            **
      ** Function: CEE5CTY - set default country    **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5CTY.

       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-5CTYQRY.
      ************************************************
      ** Call CEE5CTY with the QUERY function,
      **     and display current country code.
      ************************************************
           MOVE 2 TO FUNCTN.
           CALL 'CEE5CTY' USING FUNCTN, COUNTRY, FC.
           IF CEE000 of FC  THEN
               DISPLAY 'THE CURRENT COUNTRY CODE IS '
                   COUNTRY
           ELSE
               DISPLAY 'CEE5CTY(query) failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

       PARA-5CTYSET.
      ************************************************
      ** If the current country code is not the US,
      ** call CEE5CTY 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 'CEE5CTY' USING FUNCTN,
                                    COUNTRY, FC
               IF CEE000 of FC  THEN
                   DISPLAY 'THE NEW COUNTRY CODE IS ',
                       COUNTRY
               ELSE
                   DISPLAY 'CEE5CTY(set) failed with msg
                       Msg-No of FC UPON CONSOLE
                   STOP RUN
               END-IF
           END-IF.

           GOBACK.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5CTY                        */
 /************************************************/
 /**                                               */
 /** Function: CEE5CTY - set current country       */
 /**                                               */
 /** In this example, a call is made to the query  */
 /** function of CEE5CTY 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. */
 /**                                               */
 /************************************************/
PLI5CTY: PROC OPTIONS(MAIN);
   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL FUNCTN   INT4;
   DCL COUNTRY  CHARACTER ( 2 );
   DCL 01 FC    FEEDBACK;

   FUNCTN = 2; /* Specify 2 for the query function  */

   /* Call CEE5CTY with the query function to       */
   /*    return the current country setting         */
   CALL CEE5CTY ( FUNCTN, COUNTRY, FC );

   /* If CEE5CTY 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( 'CEE5CTY 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 CEE5CTY ( FUNCTN, COUNTRY, FC );

      /* If CEE5CTY 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( 'CEE5CTY failed with msg '
            || FC.MsgNo );
         STOP;
         END;

      END;

END PLI5CTY;
```

CEE5DLY—Suspend Processing of the Active Enclave

For COBOL/VSE and C/VSE applications, CEE5DLY provides a service similar to the PL1/VSE DELAY function. For a specified number of seconds, CEE5DLY suspends processing of the active enclave.

Syntax

interval (input)

A full-word binary value in the range of 1 to 3600, which 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE3PZ	3	—	The service has been called under CICS, or the interval supplied is not within the valid range.

Usage Notes

- This service is not available under CICS. Use instead `EXEC CICS DELAY` when running as a CICS program.
- This service is not intented for timing requests. Delays up to the nearest second may occur in some circumstances. PL/1 VSE users can alternatively use the DELAY language function to suspend enclave execution.

Examples

- **C Example**

```
 /* EDCXYDLY
    This example illustrates the use of the CEE5DLY service.
  */
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdio.h>
#include <leawi.h>
#include <ceeedcct.h>

main()
{
  _FEEDBACK fc;
  _UINT4 delay;
  time_t ltime;
  time(&ltime);
  printf("Start time :  %s", ctime(&ltime));

  delay = 5;                        /* wait for 5 seconds */

  CEE5DLY(&delay,&fc);              /* call service       */
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE5DLY failed with message number %d\n",
            fc.tok_msgno);
     exit(999);
   }
  time(&ltime);
  printf("Completion time : %s", ctime(&ltime));
}
```

- **COBOL Example**

```
CBL LIB,APOST,NOSEQ
      *                                               *
      ************************************************
      *Module/File Name: IGZTDLY
      ************************************************
      **                                            **
      ** CBLDLY - Call CEE5DLY to suspend enclave   **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    IGZTDLY.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA  DIVISION.
       WORKING-STORAGE SECTION.
***** * ************************************************
       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 delay-value                  PIC S9(9)      COMP.
       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.
           STRING FUNCTION CURRENT-DATE (5:2) '/'
                  FUNCTION CURRENT-DATE (7:2) '/'
                  FUNCTION CURRENT-DATE (3:2)
                      DELIMITED BY SIZE
                      INTO CURRENT-DATE.
           STRING FUNCTION CURRENT-DATE (9:2)  ':'
                  FUNCTION CURRENT-DATE (11:2) ':'
                  FUNCTION CURRENT-DATE (13:2)
                      DELIMITED BY SIZE
                      INTO TIME-OF-DAY.
           DISPLAY 'IGZTDLY - Begins.' UPON CONSOLE.
           DISPLAY 'IGZTDLY - On : ' CURRENT-DATE
                                     Upon Console.
           DISPLAY 'IGZTDLY - At : ' TIME-OF-DAY
                                     Upon Console.
           DISPLAY 'IGZTDLY - Calling Delay service for
                            5 secs.'.
           Move 5 to delay-value.
           CALL 'CEE5DLY' USING delay-value
                                fc.
           If NOT CEE000 of fc then
             DISPLAY 'IGZTDLY - Delay call has failed'
             Display 'IGZTDLY - Delay failed with MsgNo. '
                              Msg-no of fc
             Go To End-Run
           End-if.
           STRING FUNCTION CURRENT-DATE (5:2) '/'
                  FUNCTION CURRENT-DATE (7:2) '/'
                  FUNCTION CURRENT-DATE (3:2)
                      DELIMITED BY SIZE
                      INTO CURRENT-DATE.
           STRING FUNCTION CURRENT-DATE (9:2)  ':'
                  FUNCTION CURRENT-DATE (11:2) ':'
                  FUNCTION CURRENT-DATE (13:2)
                      DELIMITED BY SIZE
                      INTO TIME-OF-DAY.
           DISPLAY 'IGZTDLY - Delay service completed'.
           DISPLAY 'IGZTDLY - On : ' CURRENT-DATE
                                     Upon Console.
           DISPLAY 'IGZTDLY - At : ' TIME-OF-DAY
                                     Upon Console.
       End-Run.
           DISPLAY 'IGZTDLY - Completed.' UPON CONSOLE.
           GOBACK.
```

The examples shown here can also be found in ICCF Library 62.

CEE5DMP—Generate Dump

CEE5DMP generates a dump of LE/VSE and loaded LE/VSE-conforming language libraries (also known as member language libraries). Sections of the dump are selectively included, depending on options specified with the *options* parameter. Output from CEE5DMP is written to the default filename CEEDUMP, unless you specify the filename of another file by using the FNAME option of CEE5DMP. If the file CEEDUMP, or the file specified by the FNAME option, is not defined in your JCL, output from CEE5DMP is written to SYSLST.

Dumps are written only in mixed-case U.S. English. Only nested enclaves within a single process are supported.

CEE5DMP 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:

```
Exception occurred during dump processing at
nnnnnnnn
```

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 CEE5DMP is complete.

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 whether the TERMTHDACT TRACE or DUMP suboption is specified, invoke dump services and then terminate the program. You do not have to call CEE5DMP to use the dump services. Any routine can use them. To support this case, dump services are invoked as follows:

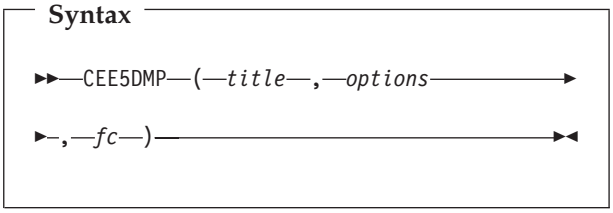
- The title is 'Condition processing resulted in the unhandled condition' to indicate why the dump was produced.
- The dump options are 'TRACE COND THR(ALL) BLOCKS STOR NOENTRY' for dump output and 'TRACE COND THR(ALL) NOBLOCK NOSTOR NOENTRY' for trace output.

Reprinting of section title and control block name at the top of each page is suppressed. Only the main title 'CEE5DMP ...' is reprinted.

The IBM-supplied default settings for the CEE5DMP options are:

```
TRACEBACK THREAD(CURRENT) FILES VARIABLES
NOBLOCKS NOSTORAGE STACKFRAME(ALL)
PAGESIZE(60) FNAME(CEEDUMP) CONDITION ENTRY
```

Syntax



title (input)
An 80-byte fixed-length character string containing a title printed at the top of each page of the dump. If longer than 60 characters, the dump title is truncated to 60 characters in order to match the record size of the dump file.

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 LE/VSE:

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.

LE/VSE supports only a single thread within an enclave. Therefore, this option is always treated as THREAD(CURRENT).

TRACEback | TRCe
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 | NOTRCe
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. In addition, file control blocks are dumped if the BLOCKS option is also 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 C or COBOL program is compiled with the TEST(SYM) compiler option. Variables and arguments are not printed for a PL/I program, regardless of the TEST compile-time option specified. The variables, arguments, and registers are dumped, beginning with the routine that called CEE5DMP. 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 LE/VSE 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 CEE5DMP. 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 run-time option is set to ON, the trace table is dumped 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 CEE5DMP, 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.

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 CEE5DMP. The first stack frame dumped is the one associated with the routine that called CEE5DMP, 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).

FNAME(filename)
Specifies the filename of the file to which the dump report is written.

The filename supplied in this option must be one of the following:

- The name of a disk file that corresponds to the name specified in a job control DLBL statement
- The name of the labeled tape file that corresponds to the name and logical unit number specified in a job control TLBL statement and job control ASSGN statement, respectively
- The logical unit number of a printer file that corresponds to the logical unit number specified in a job control ASSGN statement
- The logical unit number of an unlabeled tape file that corresponds to the logical unit number specified in a job control ASSGN statement

Table 26 lists the job control statements you should provide for each allowable type of dump file.

Table 26. Defining an I/O Device for the Dump Report File

File Type	JCL to Define File
Default (CEEDUMP)	`// DLBL CEEDUMP...`
Disk file (*filename*)	`// DLBL filename...`
Labeled tape file (SYS*nnn*)	`// TLBL SYSnnn,... // ASSGN SYSnnn,cuu`
Printer (SYS*nnn*)	`// ASSGN SYSnnn,cuu`
Unlabeled tape (SYS*nnn*)	`// ASSGN SYSnnn,cuu`

The default filename CEEDUMP is used if this option is not specified.

If the filename supplied is not valid, or the file specified by the filename is not defined in your job control, output from CEE5DMP is written to SYSLST.

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 LE/VSE-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.

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 CEE5DMP and the contents of the registers on entry to CEE5DMP.

NOENTRY
Does not include in the dump a description of the routine that called CEE5DMP and the contents of the registers on entry to CEE5DMP.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE30U	2	3102	Invalid CEE5DMP options or suboptions were found and ignored.
CEE30V	3	3103	An error occurred in writing messages to the dump file.
CEE30V	3	3103	An error occurred in writing messages to the dump file.
CEE3M6	2	3782	The value of the REGSTOR option of CEE5DMP was not in the valid range 0-256.

Usage Notes

- CICS consideration—Only ENCLAVE(CURRENT) and ENCLAVE(1) are supported under CICS.
- C consideration—In ILC applications in which a C routine calls another member language routine, and that routine in turn calls CEE5DMP, traceback information for the C routine is not provided in the dump.

For More Information

- For more information about the TERMTHDACT run-time option, see "TERMTHDACT" on page 44.
- For more information on generating dumps see *LE/VSE Debugging Guide and Run-Time Messages*.

Examples

- **C Example**

```
/*Module/File Name: EDC5DMP   */

#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");

  CEE5DMP(title,options,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE5DMP failed with msgno %d\n",
       fc.tok_msgno);
     exit(2999);
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5DMP
      ************************************************
      **                                            **
      ** CBL5DMP - Call CEE5DMP to generate a dump  **
      **                                            **
      ** In this example, a call to CEE5DMP is made **
      ** to request a dump of the run-time          **
      ** environment. Several options are specified **
      ** to customize the dump.                     **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5DMP.

       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-CBL5DMP.
           MOVE 'This is the dump report title.'
                TO DMPTITL.
           MOVE 'TRACE NOVAR SF(0) NOCOND NOENTRY'
                TO OPTIONS.
           CALL 'CEE5DMP' USING DMPTITL, OPTIONS, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY 'CEE5DMP failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

- **PL/I Example**

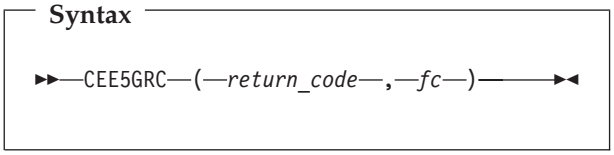
```
*PROCESS MACRO;
 /*Module/File name: IBM5DMP                        */
 /*************************************************/
 /**                                               */
 /** Function: CEE5DMP - generate dump             */
 /**                                               */
 /** In this example, a call to CEE5DMP is made to  */
 /** request a dump of the run-time environment.    */
 /** Several options are specified, to customize    */
 /** the dump.                                      */
 /**                                               */
 /**                                               */
 /*************************************************/
 PLI5DMP: PROC OPTIONS(MAIN);
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL DMPTITL  CHAR80;
    DCL OPTIONS  CHARACTER ( 255 );
    DCL 01 FC    FEEDBACK;

    /* 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 CEE5DMP with options to customize dump   */
    CALL CEE5DMP ( 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( 'CEE5DMP failed with msg '
          || FC.MsgNo );
       STOP;
```

```
      END;

END PLI5DMP;
```

CEE5GRC—Get the Enclave Return Code

CEE5GRC retrieves the current value of the user enclave return code. Use CEE5GRC in conjunction with CEE5SRC to get and then set user enclave return codes.

Syntax



return_code (output)
The enclave return code.

fc (output)
A feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

For More Information

- For more information about the CEE5SRC callable service, see "CEE5SRC—Set the Enclave Return Code" on page 94.
- For more information about the CEE5GRC and CEE5SRC callable services, see *LE/VSE Programming Guide*.

Examples

- **C Examples**

```
/*Module/File Name: EDC5GRC   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

 /*****************************************************/
 /**                                                 */
 /** Function: CEEHDLR - Register user condition     */
 /**                     handler                     */
 /**         : CEE5GRC - 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
  *****************************************************/
void CESETRC(_FEEDBACK *, _INT4*, _INT4 *, _FEEDBACK *);

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 CEE5GRC and check if enclave return code was set.
  *****************************************************/
 CEE5GRC(&enclave_RC, &feedback);
   if ( _FBCHECK ( feedback , CEE000 ) != 0 ) {
      printf("CEE5GRC failed with message number %d\n",
             feedback.tok_msgno);
      exit(2999);
   }

 if (enclave_RC != 999998)
   printf ("Error setting enclave return code");
}
/*Module/File Name: EDC5SRC   */
 /*****************************************************/
 /**                                                 */
 /** Function: CEE5SRC - Set the enclave return code. */
 /*                                                  */
 /* This is the user-written condition handler       */
 /* registered by CEGETRC. It invokes CEE5SRC 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
 /*****************************************************/

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;
   CEE5SRC(&enclave_RC, &feedback);
   *result = RESUME;
  }
 else
  {
   *result = PERCOLATE;
  }
}
```

```
/*Module/File Name: EDCDIVX   */
#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;

}
```

- **COBOL Examples**

```
CBL LIB,APOST,C,RENT,OPTIMIZE,NODYNAM
      *Module/File Name: IGZT5GRC
      ************************************************
      **                                            **
      ** CBL2GRC - Call the following LE services:    **
      **                                            **
      **         : CEEHDLR - register user condition  **
      **                     handler                **
      **         : CEE5GRC - 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.    CBL5GRC.

       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 CEE5GRC  to get the enclave return code
      ************************************************
           CALL 'CEE5GRC' 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 CBL5GRC.


CBL C,RENT,OPTIMIZE,NODYNAM,LIB,APOST
      *Module/File Name: IGZT5SRC
      ************************************************
      **                                            **
      ** DRV5SRC - Drive sample program for CEE5SRC. **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRV5SRC.

       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 'CBL5SRC'.
           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 DRV5SRC.

*************************************************
**                                             **
** CBL5SRC - Call CEE5SRC 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. CBL5SRC.

 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 (CEE549)
*************************************************
     IF CEE349 of CURRENT-CONDITION  THEN
         MOVE 761 TO ENCLAVE-RC
         CALL 'CEE5SRC' USING ENCLAVE-RC,
                              FEEDBACK
         IF NOT CEE000 of FEEDBACK  THEN
             DISPLAY 'CEE5SRC failed with msg '
                 Msg-No of FEEDBACK UPON CONSOLE
         END-IF
     END-IF.
     SET PERCOLATE TO TRUE

     GOBACK.

 END PROGRAM CBL5SRC.

CBL LIB,APOST,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.
```

- **PL/I Example**

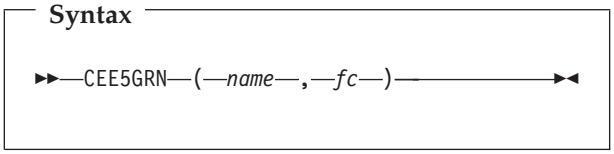
```
*Process lc(101),opt(0),s,map,list,stmt,a(f),ag,macro ;
 /*Module/File name: IBMDIV                          */
 /***************************************************/
 /**                                                 */
 /** Function: CEE5SRC - Set the enclave return code */
 /**         : CEE5GRC - 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 CEE5GRC 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;

    DCL Enclave_RC INT4;
    DCL 01 FC      FEEDBACK ;
 /***************************************************/
 /* A ZERODIVIDE ON-unit is established             */
 /***************************************************/
    on zerodivide begin;
       call CEE5GRC (Enclave_RC, fc);
       IF  FBCHECK( FC, CEE000)  THEN  DO;
          PUT SKIP LIST( 'Original Enclave RC was '
             || Enclave_RC );
          END;
       ELSE  DO;
          DISPLAY( 'CEE5GRC failed with msg '
             || FC.MsgNo );
          STOP;
```

```
         END;
      Enclave_RC = Enclave_RC + 4444;
      call CEE5SRC (Enclave_RC, fc);
      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST( 'New Enclave RC is '
            || Enclave_RC );
         END;
      ELSE  DO;
         DISPLAY( 'CEE5SRC 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;
```

CEE5GRN—Get Name of Routine that Incurred Condition

CEE5GRN gets the name of the most current LE/VSE-conforming routine where a condition occurred. If there are nested conditions, the most recently signaled condition is used.

Syntax



name (output)

A fixed-length 80-character string 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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

- **C Example**

```
/*Module/File Name: EDC5GRN   */

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
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                                      */
   CEE5GRN(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;
}
```

- **COBOL Example**

```
CBL LIB,APOST,NOOPT
      *Module/File Name: IGZT5GRN
      ************************************************
      **                                            **
      ** DRV5GRN - Drive sample program for CEE5GRN. **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRV5GRN.

       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 'CBL5GRN'.
           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 DRV5GRN.

      ************************************************
      **                                            **
      ** CBL5GRN - Call CEE5GRN to get the name of  **
      **           the routine that incurred        **
      **           the condition.                   **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5GRN.
       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-CBL5GRN.
           CALL 'CEE5GRN' USING RNAME, FC.
           IF  CEE000 of FC  THEN
               DISPLAY 'Name of routine which '
                   'incurred the condition is:  ' RNAME
           ELSE
               DISPLAY 'CEE5GRN 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 CBL5GRN.
```

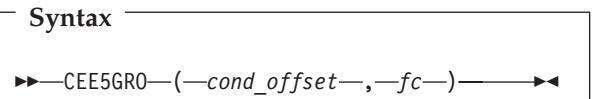
- **PL/I Example**

```
*PROCESS OPT(0), MACRO;
 /*Module/File name: IBM5GRN                */
 /******************************************/
 /**                                      **/
 /** Function: CEE5GRN - example of CEE5GRN **/
 /**                     invoked from PL/I  **/
 /**                     ON-unit            **/
 /**                                      **/
 /******************************************/

 IBM5GRN:  PROCEDURE  OPTIONS(MAIN);

  %INCLUDE CEEIBMAW;
  %INCLUDE CEEIBMCT;

  DECLARE
     RNAME       CHAR80,
     01 FC       FEEDBACK,
     divisor     FIXED BINARY(31) INITIAL(0);

  ON  ZERODIVIDE  BEGIN;

  /* Call CEE5GRN to get the name of the       */
  /* routine that incurred the most recently   */
  /* signaled condition                        */
     CALL CEE5GRN ( RNAME, FC );
     IF  FBCHECK( FC, CEE000)  THEN  DO;
        PUT SKIP LIST( 'The most recently signaled '
           || 'condition incurred by ' || RNAME );
        END;
     ELSE  DO;
        DISPLAY( 'CEE5GRN failed with msg '
           || FC.MsgNo );
        END;

     END /* ON ZeroDivide */;


  divisor = 15 / divisor /* signal ZERODIVIDE */;

END IBM5GRN;
```

CEE5GRO—Get Offset of Most Recent Condition

The CEE5GRO 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



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)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

COBOL Example of CEE5GRO Callable Service

```
CBL  LIB,QUOTE,NOOPT
      *Module/File Name: IGZT5GRO
      ************************************************
      **                                            **
      ** DRV5GRO - Register a condition handler     **
      **           that calls CEE5GRO to determine  **
      **           the offset in the program that   **
      **           incurred the condition.          **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  DRV5GRO.
       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 'CBL5GRO'.
           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 DRV5GRO.
************************************************
** CBL5GRO - Call CEE5GRO to get the offset   **
**           in the program that incurred     **
**           the condition.                   **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CBL5GRO.
 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-CBL5GRO.
     CALL 'CEE5GRO' USING ROFFSET, FC.
     IF CEE000 of FC THEN
        DISPLAY 'Offset in routine which '
                'incurred the condition is:  '
                ROFFSET
     ELSE
        DISPLAY 'CEE5GRO 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 CBL5GRO.
```

CEE5LNG—Set National Language

CEE5LNG sets the current national language. You can also use CEE5LNG to query the current national language. The national language is recorded on a LIFO national language stack. 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.

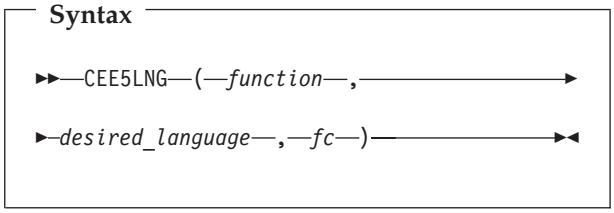
More than one national language can be pushed. Languages are popped in LIFO order. Message modules associated with a national language are not loaded until they are actually needed in order to format a message.

If you specify a *desired_language* not available on your system, the IBM-supplied default *desired_language* is used. In LE/VSE, this is UEN (uppercase U.S. English). CEEUOPT and CEEDOPT can specify an unknown national language code. They give a return code of 4 and a warning message. If an invalid language is specified, the IBM-supplied default UEN (uppercase U.S. English) is used.

You can also use the NATLANG run-time option to set the national language.

CEE5LNG affects only the LE/VSE NLS and date and time services, not the LE/VSE locale callable services or C locale-sensitive functions.

Syntax



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 CEE5LNG 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 CEE5LNG. 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 CEE5LNG 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 specified as:**	**Then** *desired_language***:**
1 or 3	Contains the desired national language identification. In this case, it is an input parameter. Table 27 on page 82 contains a list of national language identifiers. **Note:** LE/VSE supports only these national languages: **UEN** Uppercase U.S. English **ENU** Mixed-case U.S. English **JPN** Japanese.
2	Returns the current language on top of the stack. In this case, it is an output parameter.
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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 CEE5LNG. The current language was returned in the desired language parameter.
CEE3BR	3	3451	The desired language *desired-language* for the PUSH or SET function for CEE5LNG was invalid. No operation was performed.
CEE3BS	3	3452	The function *function* specified for CEE5LNG was not recognized. No operation was performed.

Usage Notes

- LE/VSE provides locales as part of the C run-time library. The locales establish the cultural conventions for locale-sensitive functions for the run-time. The CEESETL callable service and other locale callable services depend on the loaded locale (for

details of all callable services, refer to the *LE/VSE Programming Guide*, SC33-6684). To change the locale, you can use the `setlocale()` C library function or the CEESETL callable service. Locale values do not affect the settings used by the CEE5LNG callable service.

The NLS callable services (such as date and time formatting, sorting, and currency symbols) are independent of the locale. CEE5LNG only affects LE/VSE National Language Support (NLS), and date and time services. If you use both NLS services and locale callable services, the results might be unpredictable.

For More Information

- For more information about the NATLANG run-time option, see "NATLANG" on page 35.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Programming Guide*.

Examples

- **C Example**

```
/*Module/File Name: EDC5LNG   */

#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 */
  CEE5LNG(&function,lang,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE5LNG 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 */
     CEE5LNG(&function,lang,&fc);
     if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEE5LNG failed with message number %d\n",
               fc.tok_msgno);
        exit(2999);
     }
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5LNG
      ************************************************
      **                                            **
      ** CBL5LNG - Set national language            **
      **                                            **
      ** In this example, CEE5LNG is called to query **
      ** the current national language setting. If   **
      ** the setting is not mixed-case U.S. English, **
      ** CEE5LNG is called to change the setting.    **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5LNG.

       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-5LNGQRY.
      ************************************************
      ** Specify 2 for QUERY function.
      ** Call CEE5LNG to query the current
      **     national language setting
      ************************************************
           MOVE 2 TO FUNCTN.
           CALL 'CEE5LNG' USING FUNCTN, LANG, FC.
           IF CEE000 of FC  THEN
               DISPLAY 'Current National Language is: '
                   LANG
           ELSE
               DISPLAY 'CEE5LNG(query) failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

       PARA-5LNGSET.
      ************************************************
      ** If the current national language is not
      **     mixed-case U.S. English, then call
      **     CEE5LNG 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 'CEE5LNG' USING FUNCTN, LANG, FC
               IF NOT CEE000 of FC  THEN
                DISPLAY 'CEE5LNG(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.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5LNG                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEE5LNG - set national language       */
 /**                                                  */
 /** In this example, CEE5LNG is called to query the */
 /** current national language setting.  If the      */
 /** setting is not mixed case American English,     */
```

```
/** CEE5LNG is called to change the setting to that */
/**                                                */
/****************************************************/
PLI5LNG: PROC OPTIONS(MAIN);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL FUNCTN  INT4;
   DCL LANG    CHARACTER ( 3 );
   DCL 01 FC   FEEDBACK;

   FUNCTN = 2; /* Specify code to query current     */
               /* national language                 */

   /* Call CEE5LNG with function code 2 to query    */
   /*    national language                          */
   CALL CEE5LNG ( FUNCTN, LANG, FC );
   IF  FBCHECK( FC, CEE000)  THEN  DO;
      PUT SKIP LIST('The current national language is '
         || LANG );
      END;
   ELSE  DO;
      DISPLAY('CEE5LNG 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 CEE5LNG ( FUNCTN, 'ENU', FC);
      IF  ¬ FBCHECK( FC, CEE000)  THEN  DO;
         DISPLAY( 'CEE5LNG failed with msg '
            || FC.MsgNo );
         STOP;
         END;
      CALL CEE5LNG ( 2, LANG, FC);
      IF  FBCHECK( FC, CEE000)  THEN  DO;
         PUT SKIP LIST('The national language is now '
            || LANG );
         END;
      ELSE  DO;
         DISPLAY( 'CEE5LNG failed with msg '
            || FC.MsgNo );
         STOP;
         END;

      END /* Language is not ENU */;
END PLI5LNG;
```

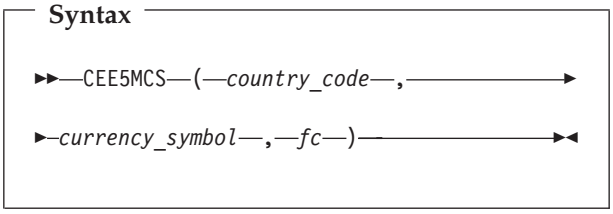
Table 27. National Language Codes

ID	National Language
AFR	Afrikaans
ARA	Arab countries
BGR	Bulgarian
CAT	Catalan
CHT	Traditional Chinese
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

CEE5MCS—Get Default Currency Symbol

CEE5MCS 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 227.

Syntax



country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 227.

country_code is not case-sensitive. If no value is specified, the default country code, as set by the COUNTRY run-time option or the CEE5CTY 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 feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3C6	2	3462	The currency symbol ' *currency-symbol* ' was truncated and was not defined in CEE5MCS.
CEE3C7	2	3463	The country code *country-code* was invalid for CEE5MCS. The default currency symbol ' *currency-symbol* ' was returned.

Usage Notes

- If you specify an invalid *country_code*, the default currency symbol is X'9F404040'.
- The returned currency symbol is only valid if it used with an applicable codeset for the specified country.

CEE5MCS and Euro Support

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 (for details, see Table 32 on page 227).

The value is taken from a typical code page for the given country. However, the actual graphical representation depends on the code page in use. 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
- Spain

As further countries pass the Economic and monetary union convergence criteria and adopt the Euro as their legal currency, the default currency symbol will replace the national currency symbol with the Euro.

For More Information

- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

```
/*Module/File Name: EDC5MCS   */

#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);
   CEE5MCS(country,currency,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE5MCS failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   printf("The default currency symbol for Canada is:"
          " %.2s\n",currency);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5MCS
      ************************************************
      **                                            **
      ** CBL5MCS - Call CEE5MCS to obtain the       **
      **           default currency symbol          **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5MCS.
       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-CBL5MCS.
      ************************************************
      ** Specify country code for the US in the call
      **     to CEE5MCS
      ************************************************
           MOVE 'US' TO COUNTRY.
           CALL 'CEE5MCS' USING COUNTRY, CURSYM, FC.

      ************************************************
      ** If CEE5MCS runs successfully, display result.
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY 'The default currency symbol '
                   'for the ' COUNTRY ' is  '' CURSYM ''''
           ELSE
               DISPLAY 'CEE5MCS failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

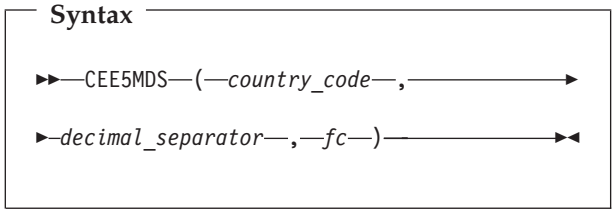
- **PL/I Example**

```
PLI5MCS: PROC OPTIONS(MAIN);
   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;
   DCL COUNTRY  CHARACTER ( 2 );
   DCL CURSYM   CHARACTER ( 4 );
   DCL 01 FC    FEEDBACK;

   COUNTRY = 'US'; /* Specify country code for the  */
                   /* United States                 */

   /* Call CEE5MCS to return currency symbol for    */
   /*    the United States                          */
   CALL CEE5MCS ( 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( 'CEE5MCS failed with msg '
         || FC.MsgNo );
      STOP;
      END;
END PLI5MCS;
```

CEE5MDS—Get Default Decimal Separator

CEE5MDS 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 227.

Syntax



country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 227.

country_code is not case-sensitive. If no value is specified, the default country code, as set by the COUNTRY run-time option or the CEE5CTY 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, indicating the service result.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 CEE5MDS.
CEE3C5	2	3461	The country code *country-code* was invalid for CEE5MDS. The default decimal separator ' *decimal-separator* ' was returned.

Usage Note

- If you specify an invalid *country_code*, the default decimal separator is a comma (,).

For More Information

- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

```
/*Module/File Name: EDC5MDS   */

#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);
   CEE5MDS(country,decimal,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE5MDS 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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5MDS
      ************************************************
      **                                            **
      ** CBL5MDS - Call CEE5MDS to get the          **
      **           default decimal separator        **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5MDS.

       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 CEE5MDS.
      ** If call was successful, print result.
      ************************************************
       PARA-CBL5MDS.
           MOVE 'US' TO COUNTRY.
           CALL 'CEE5MDS' USING COUNTRY, DECSEP, FC.
           IF CEE000 of FC  THEN
               DISPLAY 'The default Decimal Separator '
                   'for ' COUNTRY ' is '' DECSEP ''''
           ELSE
               DISPLAY 'CEE5MDS failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

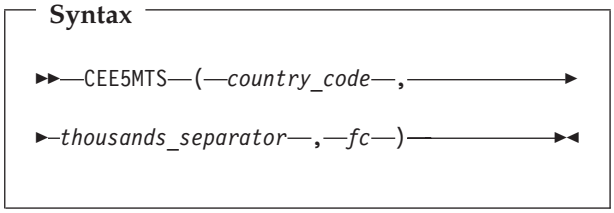
- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5MDS                      */
 /************************************************/
 /**                                             */
 /** Function: CEE5MDS - get default decimal     */
 /**           separator                         */
 /************************************************/
 PLI5MDS: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;

    %INCLUDE  CEEIBMCT;

    DCL COUNTRY  CHARACTER ( 2 );
    DCL DECSEP   CHARACTER ( 2 );
    DCL 01 FC    FEEDBACK;

    COUNTRY = 'US';   /* Specify country code for  */
                      /* the United States         */

    /* Call CEE5MDS to get default decimal         */
    /*    separator for the US                     */
    CALL CEE5MDS ( 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( 'CEE5MDS failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLI5MDS;
```

CEE5MTS—Get Default Thousands Separator

CEE5MTS returns the default thousands separator for the specified country with *country_code*.

Syntax



country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 227.

country_code is not case-sensitive. If no value is specified, the default country code, as set by the COUNTRY run-time option or the CEE5CTY 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 CEE5MTS.
CEE3C9	2	3465	The country code *country-code* was invalid for CEE5MTS. The default thousands separator ' *thousands-separator* ' was returned.

Usage Note

- If you specify an invalid *country_code*, the default thousands separator is a period (.).

For More Information

- For a list of the default settings for a specified country, see Table 32 on page 227.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

```
/*Module/File Name: EDC5MTS   */

#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);
   CEE5MTS(country,thousand,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE5MTS 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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5MTS
      ************************************************
      **                                            **
      ** CBL5MTS - Call CEE5MTS to obtain the       **
      **           default thousands separator      **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5MTS.

       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-CBL5MTS.
      ************************************************
      ** Specify the country code for the US in the
      **     call to CEE5MTS.
      ************************************************
           MOVE 'US' TO COUNTRY.
           CALL 'CEE5MTS' USING COUNTRY, THOUSEP, FC.

      ************************************************
      ** If CEE5MTS runs successfully, display result
      ************************************************
           IF CEE000 of FC  THEN
              DISPLAY 'The default Thousands Separator'
                  ' for ' COUNTRY ' is '' THOUSEP ''''
           ELSE
              DISPLAY 'CEE5MDS failed with msg '
                  Msg-No of FC UPON CONSOLE
              STOP RUN
           END-IF.

           GOBACK.
```

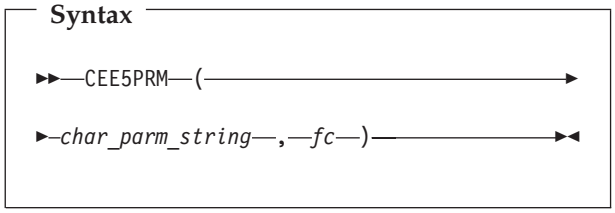
- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5MTS                          */
 /*************************************************/
 /**                                                */
 /** Function: CEE5MTS - obtain default thousands   */
 /** separator                                      */
 /**                                                */
 /*************************************************/
 PLI5MTS: PROC OPTIONS(MAIN);
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL COUNTRY  CHARACTER ( 2 );
    DCL THOUSEP  CHARACTER ( 2 );
    DCL 01 FC    FEEDBACK;

    COUNTRY = 'US';   /* Specify US as the country  */
                      /* code for the United States */
    /* Call CEE5MTS to return default thousands     */
    /*    separator for the US                      */
    CALL CEE5MTS ( COUNTRY, THOUSEP, FC );

    /* If CEE5MTS 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( 'CEE5MTS failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLI5MTS;
```

CEE5PRM—Query Parameter String

CEE5PRM 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



char_parm_string (output)
A 80-byte fixed-length string passed by CEE5PRM.

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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3I1	1	3649	The parameter string returned from CEE5PRM exceeded the maximum length of 80 bytes and was truncated.

Usage Notes

- C users can use `__osplist` to return a program argument longer than 80 characters.
- LE/VSE users can use CEE5PRML (instead of CEE5PRM) to retrieve parameters longer than 80 characters. This service is currently not available on other platforms. For details, see "CEE5PRML—Query Parameter String (Long Parameters)" on page 88.

Examples

- **C Example**

```
/*Module/File Name: EDC5PRM   */

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>

int main() {

   _CHAR80 parm;
   _FEEDBACK fc;

   CEE5PRM(parm,&fc);
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE5PRM failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
   }
   printf("%.80s\n",parm);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5PRM
      ************************************************
      **                                            **
      ** CBL5PRM - Call CEE5PRM to query the        **
      **           parameter string                 **
      **                                            **
      ** In this example, a call is made to         **
      ** CEE5PRM to return the parameter string     **
      ** that was specified at invocation of the    **
      ** program. The string is then displayed.     **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5PRM.

       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-CBL5PRM.
           CALL 'CEE5PRM' USING PARMSTR, FC.
           IF CEE000 THEN
               DISPLAY 'Program arguments specified: '
                   PARMSTR
           ELSE
               DISPLAY 'CEE5PRM failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

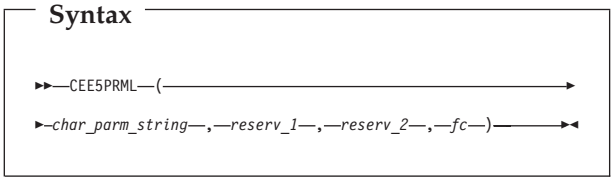
- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5PRM                          */
 /*************************************************/
 /**                                                 */
 /** Function: CEE5PRM - Query Parameter String     */
 /**                                                 */
 /*************************************************/
 PLI5PRM: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL PARMSTR CHAR80;
    DCL 01 FC FEEDBACK;
   /* Call CEE5PRM to return the program arguments  */
   /* specified at invocation of the program        */
    CALL CEE5PRM ( PARMSTR, FC );

   /* If call to CEE5PRM is successful, print       */
   /* the result                                    */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'These program arguments were specified:  "'
          || PARMSTR || '"');
       END;
    ELSE  DO;
       DISPLAY( 'CEE5PRM failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLI5PRM;
```

CEE5PRML—Query Parameter String (Long Parameters)

CEE5PRML 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



char_parm_string (output)
A 300-byte fixed-length string passed by CEE5PRML.

On return from this service, the *char_parm_string* contains the parameter string specified at invocation of the program. If the parameter string is shorter than 300 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.

reserv_1
A 2-byte field. Reserved for future use only.

reserv_2
A 4-byte field. Reserved for future use only.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

Usage Notes

LE/VSE users can use CEE5PRML (instead of CEE5PRM) to retrieve parameters longer than 80 characters. This service is currently not available on other platforms.

Examples

- **C Example**

```
/*Module/File Name: EDC5PRML   */

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>

int main() {

   _CHAR300 parm;
   _FEEDBACK fc;

   CEE5PRML(parm,0,0,&fc);
   printf("%.300s\n",parm);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5PRL
      ************************************************
      **                                            **
      ** CBL5PRML - Call CEE5PRML to query the      **
      **           parameter string                 **
      **                                            **
      ** In this example, a call is made to         **
      ** CEE5PRML to return the parameter string    **
      ** that was specified at invocation of the    **
      ** program. The string is then displayed.     **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5PRML.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  PARMSTR                PIC X(300).
       01  PARM2                  PIC X(2).
       01  PARM4                  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-CBL5PRM.
       CALL 'CEE5PRML' USING PARMSTR, PARM2, PARM4, FC.
       DISPLAY 'Program arguments specified: '
             PARMSTR
       GOBACK.
```

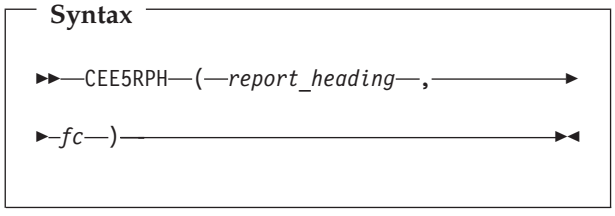
- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5PRML                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEE5PRML - Query Parameter String      */
 /**                                                  */
 /*****************************************************/
 PLI5PRML: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL PARMSTR CHAR300;
    DCL 01 FC FEEDBACK;
    DCL 01 DUMMY_LEN INT2;
    DCL 01 DUMMY_ADDR POINTER;
   /* Call CEE5PRML to return the program arguments */
   /* specified at invocation of the program        */
    CALL CEE5PRML ( PARMSTR, DUMMY_LEN, DUMMY_ADDR, FC );

   /* print the result                              */
    PUT SKIP LIST(
          'These program arguments were specified:  "'
          || PARMSTR || '"');

 END PLI5PRML;
```

CEE5RPH—Set Report Heading

CEE5RPH sets the heading displayed at the top of the storage or options report generated when you specify the RPTSTG(ON) or RPTOPTS(ON) run-time options. Figure 3 on page 39 contains a sample storage report, and Figure 2 on page 38 contains a sample options report.

Syntax



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. LE/VSE 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Note

- PL/I considerations—CEE5RPH is designed to provide an equivalent function to the special DOS PL/I variable PLIXHD.

For More Information

- For more information about the RPTSTG run-time option, see "RPTSTG" on page 38.
- For more information about the RPTOPTS run-time option, see "RPTOPTS" on page 37.
- For further information on PLIXHD, see *DOS PL/I Optimizing Compiler: Programmer's Guide*

Examples

- **C Example**

```
/*Module/File Name: EDC5RPH                        */
 /* Depending which machine this is built on, the */
 /* following option may need changing.           */
 /* pragma runopts(RPTSTG)                        */
#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 */
   CEE5RPH(heading,NULL);
  /* .
     .
     . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5RPH
      ************************************************
      **                                            **
      ** CBL5RPH - Call CEE5RPH to set report heading**
      **                                            **
      ** In this example, a call is made to CEE5RPH  **
      ** 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. CBL5RPH.

       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 CEE5RPH
      ************************************************
       PARA-CBL5RPH.
           MOVE 'My options and storage reports heading'
               TO RPTHEAD.
           CALL 'CEE5RPH' USING RPTHEAD, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY 'CEE5RPH failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5RPH                */
 /******************************************/
 /**                                      */
 /** Function: CEE5RPH - set report heading  */
 /**                                      */
 /******************************************/
 PLI5RPH: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL RPTHEAD  CHAR80;
    DCL 01 FC    FEEDBACK;
    /* Define report heading                */
    RPTHEAD = 'My storage report heading';

    /* Set report heading in call to CEE5RPH */
    CALL CEE5RPH ( RPTHEAD, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'Report heading now set to "'
          || RPTHEAD || '"' );
       END;
    ELSE  DO;
       DISPLAY( 'CEE5RPH failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLI5RPH;
```

CEE5SPM—Query and Modify LE/VSE Hardware Condition Enablement

CEE5SPM queries and modifies the enablement of LE/VSE hardware conditions. You can use the CEE5SPM service to:

- Alter the settings of the LE/VSE conditions to those specified by the caller.
- Query the current settings of the LE/VSE conditions and return the settings to the caller.
- Push the current settings of the LE/VSE conditions on to the LE/VSE-managed condition stack, where all program math settings are kept.
- Pop the pushed settings of the LE/VSE conditions from the LE/VSE-managed condition stack and set the current settings of the LE/VSE conditions to those popped.
- Push the current settings of the LE/VSE conditions on to the LE/VSE-managed condition stack and alter the settings of the LE/VSE conditions to those supplied by the caller.

The enabled or disabled LE/VSE 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.

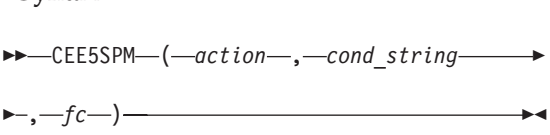
significance
: When enabled, raises the significance condition when the resulting fraction in floating-point addition or subtraction is zero.

When you use the CEE5SPM 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 CEE5SPM.

LE/VSE 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 LE/VSE feedback codes appear in Table 28 on page 92.

Syntax



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 LE/VSE conditions to those specified in the *cond_string* parameter.

2—QUERY
Queries the current settings of the LE/VSE conditions and returns the settings in the *cond_string* parameter.

3—PUSH
Pushes the current settings of the LE/VSE conditions on to the LE/VSE-managed condition stack.

4—POP
Pops the pushed settings of the LE/VSE conditions from the LE/VSE-managed condition stack and sets the current settings of the LE/VSE conditions to those popped.

5—PUSH, SET
Pushes the current settings of the LE/VSE conditions on to the LE/VSE-managed condition stack and alters the settings of the LE/VSE 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 LE/VSE 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE36V	4	3295	The condition string from CEE5SPM did not contain all of the settings, because the returned string was truncated.

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE370	4	3296	Some of the data in the condition string from CEE5SPM could not be recognized.
CEE371	4	3297	The service completed successfully for recognized condition(s), unsuccessfully for unrecognized (invalid) condition(s).
CEE372	4	3298	CEE5SPM attempted to PUSH settings onto a full stack.
CEE373	4	3299	CEE5SPM attempted to POP settings off an empty stack.
CEE374	4	3300	The action parameter in CEE5SPM was not one of the digits 1 to 5.

Usage Notes

- C consideration—C ignores the fixed-overflow, decimal-overflow, underflow, and significance interrupts, no matter what you specify in CEE5SPM.
- COBOL consideration—COBOL ignores the fixed-overflow and decimal-overflow interrupts, no matter what you specify in CEE5SPM.

Table 28. 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

- **C Example**

  You cannot use CEE5SPM to enable the fixed-overflow, decimal-overflow, underflow or significance interrupts. You can, however, query the settings of CEE5SPM.

```
/*Module/File Name: EDC5SPM   */

/*****************************************************/
/* This example queries the enablement of LE/VSE     */
/* 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;
  CEE5SPM(&action,cond_string,&fc);

  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE5SPM query failed with message %\n",
            fc.tok_msgno);
     exit(2999);
   }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5SPM
      ************************************************
      **                                            **
      ** CBL5SPM - Call CEE5SPM to query and modify  **
      **           LE hardware condition enablement  **
      **                                            **
      ** In this example, a call is made to CEE5SPM  **
      ** to check the setting of the program mask.   **
      ** See the parameter list of CEE5SPM to        **
      ** interpret what is returned as CONDSTR in    **
      ** this example.                               **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5SPM.

       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 CEE5SPM to return
      **     the condition string DISPLAY results.
      ************************************************
           PARA-CBL5SPM.
               MOVE 2 TO ACTION.
               CALL 'CEE5SPM' USING ACTION, CONDSTR, FC.
               IF  CEE000 of FC  THEN
                   DISPLAY 'The current setting of the ',
                       'program mask is: ' CONDSTR
               ELSE
                   DISPLAY 'CEE5SPM failed with msg '
                       Msg-No of FC UPON CONSOLE
                   STOP RUN
               END-IF.

               GOBACK.
```

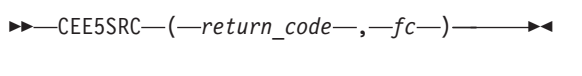
- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5SPM                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEE5SPM - Query and Modify LE/VSE     */
 /**                   Hardware Condition Enablement */
 /**                                                  */
 /** This example calls CEE5SPM to query the current */
 /** setting of the program mask. See the parameter  */
 /** list of CEE5SPM to interpret what is returned   */
 /** as CONDSTR in this example.                     */
 /**                                                  */
 /*****************************************************/
 PLI5SPM: PROC OPTIONS(MAIN);
   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL ACTION   INT4;
   DCL CONDSTR  CHAR80;
   DCL 01 FC    FEEDBACK;

   /* Call CEE5SPM to query the current setting of */
   /*    the program mask                          */
   ACTION = 2  /* Specify action code 2 to query   */
               /* the program mask                */;
   CALL CEE5SPM ( 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( 'CEE5SPM failed with msg ' || FC.MsgNo );
      STOP;
      END;

   /* Call CEE5SPM to enable specification exceptions */
   ACTION = 1  /* Specify action code 1 to SET the    */
               /*    program mask                     */;
   CONDSTR = 'S'  /* Specify a program mask that will */
                  /*   allow specification exceptions */
                  /*   (all others remain unchanged) */;
   CALL CEE5SPM ( ACTION, CONDSTR, FC );
   IF  ¬ FBCHECK( FC, CEE000)  THEN  DO;
      DISPLAY( 'CEE5SPM failed with msg ' || FC.MsgNo );
      STOP;
      END;

   CALL CEE5SPM ( 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( 'CEE5SPM failed with msg ' || FC.MsgNo );
      STOP;
      END;

 END PLI5SPM;
```

CEE5SRC—Set the Enclave Return Code

CEE5SRC sets the user enclave return code. The value set is used in the calculation of the final enclave return code at enclave termination.

Syntax



return_code(input)
The enclave return code to be set should be <= 999,999 and >= 0 to be in the LE/VSE-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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE0HA	1	0554	A value outside the range of 0 through 999,999 was supplied. However, the value was still used as the enclave return code.

For More Information

- For more information about the CEE5SRC callable service, see the *LE/VSE Programming Guide*.

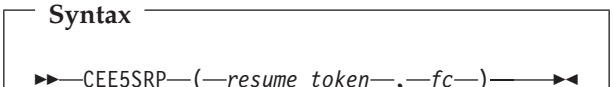
Examples

CEE5SRC is used with CEE5GRC—see the examples for CEE5GRC, "Examples" on page 73.

CEE5SRP—Set Resume Point

The CEE5SRP service sets the resume point at the next instruction in the calling routine. CEE5SRP works only in conjunction with the CEEMRCE service.

Syntax



resume_token (output)
An INT4 data type that, upon completion of this service, contains a token that represents a machine state block, which LE/VSE allocates from heap storage. LE/VSE 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)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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

- Use the CEE5SRP service only with the CEEMRCE service from a user condition handler. The token returned by this service is used as input to the CEEMRCE service.
- LE/VSE 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.

For More Information

For details of how the CEEMRCE and CEE5SRP callable services are used, refer to the chapter "LE/VSE Condition Handling Introduction" in the *LE/VSE Programming Guide*.

Examples

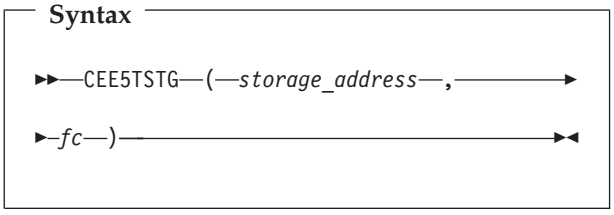
For examples of how to use CEE5SRP in comnination with CEEMRCE and CEEHDLR, see "COBOL Example of CEEMRCE Callable Service" on page 165.

CEE5TSTG—Test Access Authority for a Supplied Storage Address

CEE5TSTG allows a programmer to test for the access that is available to a specified storage address. The service returns a feedback token indicating what access is available to the supplied storage address. This can be either:

- no access permitted in current execution key.
- read-only access.
- update access.

Syntax



storage_address (input)
A fullword binary integer that indicates the address that is to be tested for access permissions.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3PZ	1	3907	No access is available to the supplied storage area.
CEE3Q0	1	3908	Update Access not available. Read access is available.
CEE3Q1	1	3909	TRAP run-time option is not set to ON. Access permission unknown. For more detailed information, refer to the associated message number.

Usage Notes

- CEE5TSTG is available in both the BATCH and CICS environments. However, it requires TRAP(ON) be set in the active enclave where CEE5TSTG is to be used.
- CEE5TSTG can be called by LE-conforming assembler main and subroutine programs.
- CEE5TSTG does not switch or change execution keys. The test is performed in the currently set execution key only.
- For read- and update-tests only, the first 4 bytes of the specified address are validated for access permissions.

Examples

- **C Example**

```
/*Module/File Name: EDCTSTG   */

#include <stdio.h>
#include <ceeedcct.h>
#include <leawi.h>

int main() {

 /*************************************************/
 /**                                             **/
 /** Function: CEE5TSTG - Test access to a specified**/
 /**                     storage address.        **/
 /**                                             **/
 /** In this example, CEE5TSTG is called to test    **/
 /** access permission to low-core.              **/
 /**                                             **/
 /*************************************************/
   _INT4 test_addr;
   _FEEDBACK fc;

   printf("EDCTSTG Begins\n");
   test_addr = malloc(1000);
   printf("Testing access for storage addr %x\n",
         test_addr);
   CEE5TSTG(&test_addr,&fc);
   if ( _FBCHECK ( fc , CEE000 ) == 0 ) {
     printf("All access permitted to %x\n",
           test_addr);
   }
   else {
     if ( _FBCHECK ( fc , CEE3Q1 ) == 0) {
        printf("EDCTSTG - Incompatible trap option ");
        printf("set See Msg CEE%d\n",fc.tok_msgno);
     }
     else {
        printf("EDCTSTG - Access specified ");
        printf("in Msg CEE%d\n",fc.tok_msgno);
     }
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST,NOSEQ
      *Module/File Name: IGZTSTG
      ************************************************
      **                                            **
      **   Call CEE5TSTG to test a storage address  **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    IGZTSTG.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA  DIVISION.
```

```
       WORKING-STORAGE SECTION.
***** * ********************************************
       01 Test-Addr               PIC S9(9) COMP.
       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.
           DISPLAY 'IGZTSTG - Begins.' UPON CONSOLE.
           DISPLAY 'IGZTSTG - Testing for Access'
                   ' to low-core storage.'.
           MOVE Zero to Test-Addr.
           CALL 'CEE5TSTG' USING Test-Addr, fc.
           IF CEE3Q1 of fc THEN
             DISPLAY 'IGZTSTG - Incompatiable TRAP '
                      'option set.'
             DISPLAY '          See Msg CEE '
                     Msg-No of fc
             GO TO End-Run
           END-IF.
           IF NOT CEE000 of fc THEN
            DISPLAY 'IGZTSTG - Access Specified in '
                    Msg-no of fc
            DISPLAY '          only permitted '
                    'to Addr ' Test-Addr
            GO TO End-Run
           END-IF.
           DISPLAY 'IGZTSTG - All Access '
                   'permitted to ' Test-Addr.
       END-RUN.
           MOVE 0 TO RETURN-CODE.
           DISPLAY 'IGZTSTG - Completed.'
                   UPON CONSOLE.
           GOBACK.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBMTSTG                        */

 /************************************************/
 /**                                            **/
 /** Function: CEE5TSTG - Test access to a      **/
 /**                    specified storage address.**/
 /**                                            **/
 /** In this example, CEE5TSTG is called to test **/
 /** access permission to low-core and then to   **/
 /** user accessable storage.                    **/
 /**                                            **/
 /************************************************/

 PLITSTG: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL REDEF_PTR          FIXED BIN(31,0)
                              BASED(ADDR(MY_POINTER));
    DCL REDEF_DIS          PIC'99999999';
    DCL MY_POINTER         POINTER;
    DCL MY_MSGNO           PIC'9999';
    DCL outchar            char(8);
    DCL 01 FC              FEEDBACK;

    REDEF_PTR = 0;       /* storage address to test */

    /* Call CEE5TSTG to test what access is         */
    /*    permitted to this address.                */
    CALL CEE5TSTG (MY_POINTER, FC );

    /* If CEE5TSTG ran successfully, print result   */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'All access is available to address ' ||
          REDEF_PTR);
       END;
    ELSE  DO;
       MY_MSGNO = FC.Msgno;
       REDEF_DIS = REDEF_PTR;
       PUT SKIP LIST(
       'CEE5TSTG indicates access specified by '
       || 'msg.no CEE'
          ||MY_MSGNO
             ||' is allowed to storage addr '
                ||HEXPTR(MY_POINTER));
       END;

    /* storage address to test */
    MY_POINTER = Addr(REDEF_PTR);

    /* Call CEE5TSTG to test what access is         */
    /*    permitted to this address.                */
    CALL CEE5TSTG (MY_POINTER, FC );

    /* If CEE5TSTG ran successfully, print result   */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST(
          'All access is available to address ' ||
          HEXPTR(MY_POINTER));
       END;
    ELSE  DO;
       MY_MSGNO = FC.Msgno;
       REDEF_DIS = REDEF_PTR;
       PUT SKIP LIST(
       'CEE5TSTG indicates access specified by '
       || 'msg.no CEE'
          ||MY_MSGNO
             ||' is allowed to storage addr '
                ||HEXPTR(MY_POINTER));
       END;

/*-------------------------------------------------*/
/* hexptr: convert a pointer field to 8 hex digits */
/*-------------------------------------------------*/
hexptr: proc(inptr) returns(char(8));
dcl   inptr    pointer,
      outchar  char(8);
dcl   charnum  bin fixed(31);
dcl   ct(0:15) static char(1)
      init(('0'),('1'),('2'),('3'),('4'),('5'),('6'),
           ('7'),('8'),('9'),('A'),('B'),('C'),('D'),
           ('E'),('F'));
dcl  (i,j)     bin fixed(31);
dcl  (repeat, unspec, substr) builtin;

j = 0;
do i = 1 to 29 by 4;
   j = j + 1;
   charnum = 0;
   unspec(charnum) = repeat('0'b, 27)
      || substr(unspec(inptr),i,4);
   substr(outchar,j,1) = ct(charnum);
   end;

return (outchar);
end hexptr; /*-------------------------------------*/
 END PLITSTG;
```

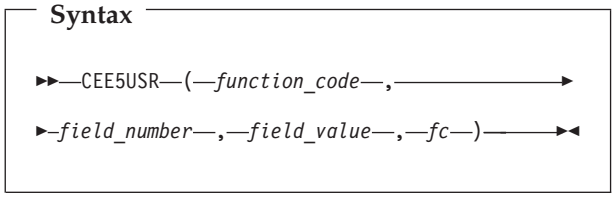
CEE5USR—Set or Query User Area Fields

CEE5USR 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 LE/VSE 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.

LE/VSE initializes both user area fields to X'00000000' during enclave initialization.

Syntax



function_code(input)
A fullword binary integer representing the function performed:

1—SET
Set the user area field according to the value specified in *field_value*.

2—QUERY
Returns the current value of the user area field 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3PS	3	3900	The function code passed to CEE5USR was not 1 or 2.
CEE3PT	3	3901	The field number passed to CEE5USR was not 1 or 2.

Examples

- **C Example**

```
/*Module/File Name: EDC5USR   */

#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;

  CEE5USR(&function_code,&field_number,&field_value,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
  printf("CEE5USR failed with message number %d\n",
          fc.tok_msgno);
     exit(2999);
  }
  /* .
     .
     . */
  /* get the value of field 2 */
  function_code = 2;
  field_number = 1;

  CEE5USR(&function_code,&field_number,&field_value,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf("CEE5USR failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
  /* .
     .
     . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZT5USR
      ************************************************
      **                                            **
      ** CBL5USR - Call CEE5USR to set or query user **
      **           area fields                      **
      **                                            **
      ** In this example, CEE5USR is called twice:  **
      ** once to set the value of a user area, and  **
      ** once to query it.                          **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL5USR.

       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-5USRSET.
           MOVE 1  TO FUNCODE.
           MOVE 1  TO FIELDNO.
           MOVE 23 TO INVALUE.
           CALL 'CEE5USR' USING FUNCODE, FIELDNO,
                                INVALUE, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY 'CEE5USR 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-5USRQRY.
           MOVE 2 TO FUNCODE.
           MOVE 1 TO FIELDNO.
           CALL 'CEE5USR' USING FUNCODE, FIELDNO,
                                INVALUE, FC.
           IF CEE000 of FC  THEN
               DISPLAY 'User Area field ' FIELDNO
                   ' is: ' INVALUE
           ELSE
               DISPLAY 'CEE5USR failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBM5USR                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEE5USR - set/query user area fields  */
 /**                                                  */
 /** In this example, CEE5USR is called twice: once  */
 /** to set the value of a user area, and once to    */
 /** query it.                                        */
 /*****************************************************/
 PLI5USR: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
    DCL FUNCODE   INT4;
    DCL FIELDNO   INT4;
    DCL OUTVALUE  INT4;
    DCL INVALUE   INT4;
    DCL 01 FC     FEEDBACK;

    FUNCODE = 1; /* Specify 1 for the set function   */
    FIELDNO = 1; /* Specify field 1 of two           */
    INVALUE = 5; /* Value to put in field 1          */
    /* Call CEE5USR to set user field 1 to 5         */
    CALL CEE5USR ( FUNCODE, FIELDNO, INVALUE, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'LE/VSE User field ' || FIELDNO
          || ' has been set to ' || INVALUE );
       END;
    ELSE  DO;
       DISPLAY( 'CEE5USR failed with msg '
          || FC.MsgNo );
       STOP;
       END;

    /* Call CEE5USR to query the value of field 1    */

    FUNCODE = 2; /* Specify 2 for query function     */
    FIELDNO = 1; /* Specify field 1 of two           */

    CALL CEE5USR ( FUNCODE, FIELDNO, OUTVALUE, FC );
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST( 'LE/VSE User field ' || FIELDNO
          || ' is currently set to ' || OUTVALUE );
       END;
    ELSE  DO;
       DISPLAY( 'CEE5USR failed with msg '
          || FC.MsgNo );
       STOP;
       END;

 END PLI5USR;
```

CEECBLDY—Convert Date to COBOL Integer Format

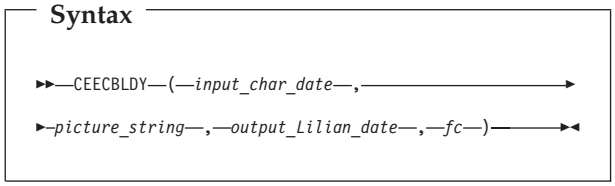
CEECBLDY converts a string representing a date into a COBOL Integer format, which is the number of days since 31 December 1600. This service is similar to CEEDAYS, except that it provides a string in COBOL Integer format, which is compatible with ANSI intrinsic functions. Use CEECBLDY to access the century window of LE/VSE and to perform date calculations with ANSI intrinsic functions.

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 LE/VSE 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 LE/VSE 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 perform calculations on dates earlier than 1 January 1601, add 4000 to the year in each date, convert the dates to COBOL Integer format, then do the calculation. If the result of the calculation is a date, as opposed to a number of days, convert the result to a date string and subtract 4000 from the year.

By default, 2-digit years lie within the 100-year range starting 80 years prior to the system date. Thus, in 1996, all 2-digit years represent dates between 1916 and 2015, inclusive. You can change this default range by using the CEESCEN callable service.

Syntax



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 233 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:

```
MOVE '6/2/88' TO DATEVAL-STRING.
MOVE 6 TO DATEVAL-LENGTH.
MOVE 'MM/DD/YY' TO PICSTR-STRING.
MOVE 8 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.

MOVE '06/02/88' TO DATEVAL-STRING.
MOVE 8 TO DATEVAL-LENGTH.
MOVE 'MM/DD/YY' TO PICSTR-STRING.
MOVE 8 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.

MOVE '060288' TO DATEVAL-STRING.
MOVE 6 TO DATEVAL-LENGTH.
MOVE 'MMDDYY' TO PICSTR-STRING.
MOVE 6 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.

MOVE '88154' TO DATEVAL-STRING.
MOVE 5 TO DATEVAL-LENGTH.
MOVE 'YYDDD' TO PICSTR-STRING.
MOVE 5 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.
```

would each assign the same value, 141502 (02 June 1988), to *COBINTDTE*.

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 233 for a list of valid picture character terms and Table 34 on page 234 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 234 for an additional example.

If *picture_string* includes a Chinese Era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *input_char_date* is replaced by the year number within the Chinese Era. For example, the year 1988 equals the Chinese year 77 in the MinKow Era. See Table 34 on page 234 for an additional example.

If *picture_string* includes a Japanese or Chinese Era symbol, the date supplied must fall within the specified Era (see Table 35 on page 234 or Table 36 on page 235 for details).

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.

CEECBLDY returns a non-CEE000 symbolic feedback code if:

1. *picture_string* includes a Japanese or Chinese Era symbol.

2. *input_char_date* contains a date which is *outside* the Era range of the specified Era *but is otherwise valid*.

output_Integer_date is calculated from the start of the specified Era. Otherwise, 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Japanese or Chinese 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 Japanese (<JJJJ>) or Chinese (<CCCC>) year-within-Era value passed to CEEDAYS or CEESECS was zero.

Usage Note

- 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.

For More Information

- For more information about the CEESCEN callable service, see "CEESCEN—Set the Century Window" on page 182.
- For a list of Japanese Eras supported by CEECBLDY, see Table 35 on page 234.
- For a list of Chinese Eras supported by CEECBLDY, see Table 36 on page 235.

Examples

- **COBOL Example**

```
CBL LIB,APOST
*Module/File Name: IGZTCBLD
************************************************
**                                            **
** Function: Invoke CEECBLDY callable service  **
** to convert date to COBOL Integer format.    **
** This service is used when using the         **
** Language Environment 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 Mmmmmmmmmmmmmz 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 Lilian day: ' INTEGER
     ELSE
         DISPLAY 'CEEBLDY failed with msg '
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

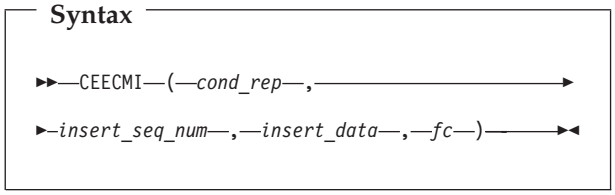
CEECMI—Store and Load Message Insert Data

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 run-time option.

ISIs are released when the value specified in the MSGQ run-time 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



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 utility.

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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.
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.

For More Information

- For more information about the MSGQ run-time option, see "MSGQ" on page 34.

- For more information about CEEBLDTX, see *LE/VSE Programming Guide*.

Examples

- **C Example**

```
/*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 LE message services to store and        *
    **   load message insert data.                   *
    **   The resulting message and insert is written *
    **   to the LE 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 CEEBLDTX 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);

}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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 LE 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.
```

- **PL/I Example**

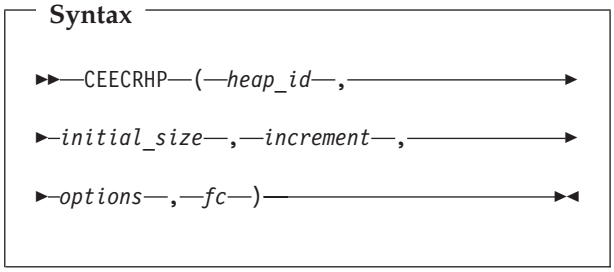
```
*PROCESS MACRO;
   /*Module/File Name: IBMCMI                          */
   /***************************************************
    **                                                 *
    ** FUNCTION  : CEECMI  - store and load message    *
    **                      insert data                *
    **           : CEEMSG  - retrieve, format, and     *
    **                      dispatch a message to      *
    **                      message file               *
    **                                                 *
    **   This example illustrates the invocation of    *
    **   LE/VSE message services to store and load     *
    **   message insert data. The resulting message    *
    **   and insert are written to the MSGFILE.        *
    **                                                 *
    ***************************************************/
 IBMCMI: Proc Options(Main);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DECLARE INSERT     VSTRING;
    DECLARE 01 CTOK    FEEDBACK;

    DECLARE 01 FBCODE  FEEDBACK;
    DECLARE MSGFILE    INT4;
    DECLARE INSERT_NO  INT4;

    /***************************************************/
    /* Ctok is initialized in the following code     */
    /* 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.               */
    /***************************************************/
    CTOK.MsgSev = 0;
    CTOK.MsgNo = 10;
    CTOK.Flags.Case = '01'B;
    CTOK.Flags.Severity = '000'B;
    CTOK.Flags.Control = '000'B;
    CTOK.FacID = 'XMP';
    CTOK.ISI = 0;

    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 LE/VSE is limited only by the amount of virtual storage available.

Syntax



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.

In the batch environment, storage obtained using *options* 79 or 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 run-time option is used. If no HEAP run-time option was provided and *initial_size* is specified as 0, CEECRHP uses the installation 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 run-time 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 29.

Table 29. 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 run-time option)
01	HEAP(,,,FREE) (location inherited from HEAP run-time option)
70	HEAP(,,,KEEP) (location inherited from HEAP run-time 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 run-time option)
76	HEAP(,,BELOW,) (disposition inherited from the HEAP run-time 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 (CEECRHP) request was invalid.
CEE0P5	3	0805	The increment size value supplied in a create heap (CEECRHP) request was invalid.
CEE0P6	3	0806	The options value supplied in a create heap (CEECRHP) request was unrecognized.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage (CEECZST) request.

For More Information

- For more information about the CEEDSHP callable service, see "CEEDSHP—Discard Heap" on page 119.
- For more information about the HEAP run-time option and IBM-supplied defaults, see "HEAP" on page 30.

Examples

- **C Example**

```
/*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);
  }
/*  .
    .
    .  */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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 run-time 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.
```

- **PL/I Example**

```
*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  INT4 ;
    DCL HPSIZE  INT4 ;
    DCL INCR    INT4 ;
    DCL OPTS    INT4 ;
    DCL 01 FC   FEEDBACK ;

    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 lesser 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.

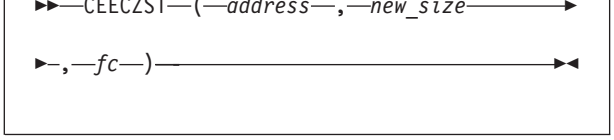
Because the new storage might be allocated at a different location from the existing allocation, any pointers (specifically any machine addresses) that referred to the old storage become invalid. Continued use of such dangling pointers gives unpredictable, and almost certainly incorrect, results.

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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 (CEEGTST) or a reallocate request (CEECZST) was not a positive number.
CEE0PA	3	0810	The storage address in a free storage (CEEFRST) request was not recognized, or heap storage (CEECZST) control information was damaged.
CEE0PD	3	0813	Insufficient storage was available to satisfy a get storage (CEECZST) request.

Usage Note

- 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.

For More Information

- For more information about the STORAGE run-time option, see "STORAGE" on page 42.
- For information about CEEGTST, see "CEEGTST—Get Heap Storage" on page 143.

Examples

- **C Example**

```
/*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);
  }
 /* .
   .
   .  */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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                PIC S9(9) BINARY.
       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.
```

- **PL/I Example**

```
*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   INT4 ;
    DCL STGSIZE  INT4 ;
    DCL ADDRSS1  POINTER;
    DCL 01 FC1   FEEDBACK ;

    DCL ADDRSS2  POINTER;
    DCL NEWSIZE  INT4 ;
    DCL 01 FC2   FEEDBACK ;

    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;
```

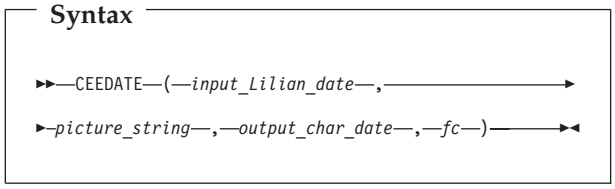
CEEDATE—Convert Lilian Date to Character Format

CEEDATE converts a number representing a Lilian date to a date written in character format. The output is a character string, such as `1996/04/23`.

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 run-time option or CEE5CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax



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 as is to *output_char_date*.

See Table 33 on page 233 for a list of valid picture characters, and Table 34 on page 234 for examples of valid picture strings.

If *picture_string* is null or blank, CEEDATE calls the CEEFMDA callable service to get *picture_string* based on the current value of the COUNTRY run-time option. For example, if the current value of the COUNTRY run-time 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 `DD.MM.YYYY`, for instance, `21.04.1993`. 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 234 for an additional example.

If *picture_string* includes a Chinese Era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *output_char_date* is replaced by the year number within the Chinese Era. For example, the year 1988 equals the Chinese year 77 in the MinKow Era. See Table 34 on page 234 for an additional example.

output_char_date (output)
A fixed-length 80-character string that is the result of converting *input_Lilian_date* to the format specified by *picture_string*. See Table 30 on page 110 for sample output dates. If *input_Lilian_date* is invalid, *output_char_date* is set to all blanks and 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.
CEE2EM	3	2518	An invalid picture string was specified in a call to a date/time service.

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE2EQ	3	2522	Japanese (<JJJJ>) or Chinese (<CCCC> or <CCCCCCCC>) Era 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.
CEE2F6	1	2534	Insufficient field width was specified for a month or weekday name in a call to CEEDATE or CEEDATM. Output set to blanks.

Usage Note

- 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.

For More Information

- For more information about the CEEDAYS callable service, see "CEEDAYS—Convert Date to Lilian Format" on page 113.
- For more information about the COUNTRY run-time option, see "COUNTRY" on page 26.
- For information about how to get the default format for a given country code, see "CEEFMDA—Get Default Date Format" on page 123.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Programming Guide*.
- For a list of Japanese Eras supported by CEEDATE, see Table 35 on page 234.
- For a list of Chinese Eras supported by CEEDATE, see Table 36 on page 235.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*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  INT4 ;
    DCL PICSTR  VSTRING;
    DCL CHRDATE CHAR80 ;
    DCL 01 FC   FEEDBACK ;

    LILIAN = 152385;  /* input date in Lilian format */
    /* picture string that describes how converted   */
    /* date is to be formatted                       */
    PICSTR = 'ZD Mmmmmmmmmmmmmmz YYYY';

    /* Call CEEDATE 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 30 shows the sample output from CEEDATE.

Table 30. 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	*MinKow* 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
	YYDDD	88140
	YY.DDD	88.140
	YYYY.DDD	1988.140
148142	YY/MM/DD	88/05/20
	YYYY/ZM/ZD	1988/5/20
148143	WWW., MMM DD, YYYY	SAT., MAY 21, 1988
	Www., Mmm DD, YYYY	Sat., May 21, 1988
	Wwwwwwwwww, Mmmmmmmmm DD, YYYY	Saturday , May 21, 1988
	Wwwwwwwwwz, Mmmmmmmmmz DD, YYYY	Saturday, May 21, 1988

CEEDATM—Convert Seconds to Character Timestamp

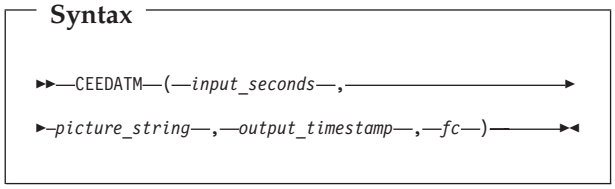
CEEDATM converts a number representing the number of seconds since 00:00:00 14 October 1582 to a character string format. The format of the output is a character string timestamp, for example: `1988/07/26 20:37:00`.

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 run-time option or

CEE5CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax



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 233 for a list of valid picture character terms and Table 34 on page 234 for examples of valid picture strings.

If *picture_string* is null or blank, CEEDATM calls the CEEFMDT callable service to get *picture_string* based on the current value of the COUNTRY run-time option. For example, if the current value of the COUNTRY run-time 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,9`".

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 234 for an example.

If *picture_string* includes the Chinese Era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *output_timestamp* represents the year within Chinese Era. See Table 34 on page 234 for an example.

output_timestamp (output)
A fixed-length 80-character string 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 31 on page 113 for sample output.

If *input_seconds* is invalid, *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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.
CEE2EA	3	2506	Japanese (<JJJJ>) or Chinese (<CCCC> or <CCCCCCCC>) Era 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 in a call to a date/time service.
CEE2EV	2	2527	The timestamp string returned by CEEDATM was truncated.
CEE2F6	1	2534	Insufficient field width was specified for a month or weekday name in a call to CEEDATE or CEEDATM. Output set to blanks.

Usage Note

- 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.

For More Information

- For more information about the CEESECS callable service, see "CEESECS—Convert Timestamp to Seconds" on page 188.
- For more information about the COUNTRY run-time option, see "COUNTRY" on page 26.
- For information about how to get a default timestamp for a given country code, see "CEEFMDT—Get Default Date and Time Format" on page 124.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Programming Guide*.
- For a list of Japanese Eras supported by CEEDATM, see Table 35 on page 234.
- For a list of Chinese Eras supported by CEEDATM, see Table 36 on page 235.

Examples

- **C Example**

```
/*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,
   "Mmmmmmmmmmz 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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*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 FLOAT8;
    DCL PICSTR  VSTRING;
    DCL TIMESTP CHAR80;
    DCL 01 FC   FEEDBACK;
    SECONDS = 13166064060; /* Input is Lilian seconds*/

    PICSTR = 'ZD Mmmmmmmmmmmmmz 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 31 shows the sample output of CEEDATM.

Table 31. Sample Output of CEEDATM

input_seconds	picture_string	output_timestamp
12,799,191,601.000	YYMMDD	880516
	HH:MI:SS	19:00:01
	YY-MM-DD	88-05-16
	YYMMDDHHMISS	880516190001
	YY-MM-DD HH:MI:SS	88-05-16 19:00:01
	YYYY-MM-DD HH:MI:SS	
	AP	1988-05-16 07:00:01 PM
12,799,191,661.986	DD Mmm YY	16 May 88
	DD MMM YY HH:MM	16 MAY 88 19:01
	WWW, MMM DD, YYYY	MON, MAY 16, 1988
	ZH:MI AP	7:01 PM
	Wwwwwwwwwz, ZM/ZD/YY	
	HH:MI:SS.99	Monday, 5/16/88 19:01:01.98
12,799,191,662.009	YYYY	1988
	YY	88
	Y	8
	MM	05
	ZM	5
	RRRR	Vbbb
	MMM	MAY
	Mmm	May
	Mmmmmmmmmm	Maybbbbbb
	Mmmmmmmmmz	May
	DD	16
	ZD	16
	DDD	137
	HH	19
	ZH	19
	MI	01
	SS	02
	99	00
	999	009
	AP	PM
	WWW	MON
	Www	Mon
	Wwwwwwwwww	Mondaybbbb
	Wwwwwwwwwz	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.

Do not use CEEDAYS in combination with COBOL intrinsic functions. Use CEECBLDY for COBOL programs that use intrinsic functions.

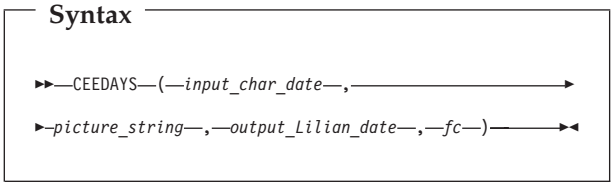
CEEDAYS can perform arithmetic on dates, such as calculating the number of days between two dates. The inverse of CEEDAYS is CEEDATE,

which converts *output_Lilian_date* from Lilian format to character format.

To perform calculations on dates earlier than 15 October 1582, add 4000 to the year in each date, convert the dates to Lilian, then do the calculation. If the result of the calculation is a date, as opposed to a number of days, convert the result to a date string and subtract 4000 from the year.

By default, 2-digit years lie within the 100-year range starting 80 years prior to the system date. Thus, in 1996, all 2-digit years represent dates between 1916 and 2015, inclusive. This default range is changed by using the callable service CEESCEN.

Syntax



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 233 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:

```
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);
```

would each assign the same value, 148155 (02 June 1988), to *lildate*.

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 233 for a list of valid picture character terms, and Table 34 on page 234 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 234 for an additional example.

If *picture_string* includes a Chinese Era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *input_char_date* is replaced by the year number within the Chinese Era. For example, the year 1988 equals the Chinese year 77 in the MinKow Era. See Table 34 on page 234 for an additional example.

If *picture_string* includes a Japanese or Chinese symbol, the date supplied must fall within the specified Era (see Table 35 on page 234 or Table 36 on page 235 for details).

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.

CEEDAYS returns a non-CEE000 symbolic feedback code if:

1. *picture_string* includes a Japanese or ROC Era symbol.
2. *input_char_date* contains a date which is outside the Era range of the specified Era *but is otherwise valid*.

output_Lilian_date is calculated from the start of the specified Era. Otherwise, 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Japanese or Chinese 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 Japanese (<JJJJ>) or Chinese (<CCCC>) year-within-Era value passed to CEEDAYS or CEESECS was zero.

Usage Note

- 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.

For More Information

- For more information about the CEEDATE callable service, see "CEEDATE—Convert Lilian Date to Character Format" on page 108.
- For more information about the CEESCEN callable service, see "CEESCEN—Set the Century Window" on page 182.
- For a list of Japanese Eras supported by CEEDATM, see Table 35 on page 234.
- For a list of Chinese Eras supported by CEEDATM, see Table 36 on page 235.

Examples

- **C Example**

```
/*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,"Mmmmmmmmmmz 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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

```
*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  VSTRING;
    DCL CHRD2    VSTRING;
    DCL PICSTR   VSTRING;
    DCL PICST2   VSTRING;
    DCL LILIAN   INT4;
    DCL LIL2     INT4;
    DCL 01 FC    FEEDBACK;

    /* 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;
```

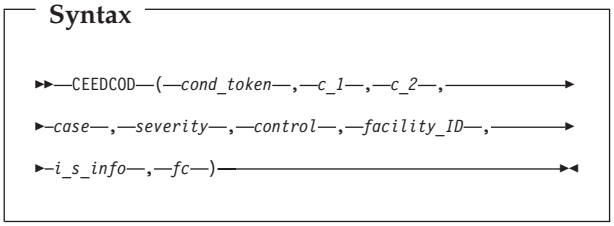
- **PL/I Example**

CEEDCOD—Decompose a Condition Token

CEEDCOD decomposes or alters an existing condition token.

LE/VSE-conforming HLLs can decompose or alter the condition token fields without using the CEEDCOD service. See the CEESGL HLL examples starting on page 194 for examples of how to alter the condition token field.

Syntax



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.

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 an LE/VSE 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Note

- C considerations—The structure of the condition token (type_FEEDBACK) is described

in the `leawi.h` header file shipped with LE/VSE. 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 175.

For More Information

- For more information about the condition_ID, facility_ID, and case 1 and case 2 conditions, see "CEENCOD—Construct a Condition Token" on page 173.
- For examples of how to alter the condition token field, see the CEESGL HLL examples starting on page "Examples" on page 194.
- 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 193.

Examples

- **C Example**

```
/*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");
  /* .
     .
     . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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 LE service, 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 LE service 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.
```

- **PL/I Example**

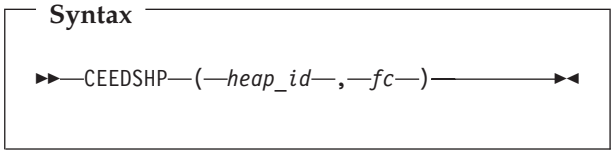
```
*PROCESS MACRO;
 /*Module/File name: IBMDCOD                          */
 /*************************************************/
 /**                                               **/
 /** Function: CEEDCOD - decompose a condition     **/
 /**                     token                     **/
 /**                                               **/
 /** 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/VSE     **/
 /** service. The condition token returned by      **/
 /** CEEGTST is used as input to CEEDCOD.          **/
 /**                                               **/
 /*************************************************/
 PLIDCOD: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
    DCL HEAPID  INT4;
    DCL STGSIZE INT4;
    DCL ADDRSS  POINTER;
    DCL 01 FC   FEEDBACK;
    DCL SEV     INT2;
    DCL MSGNO   INT2;
    DCL CASE    INT2;
    DCL SEV2    INT2;
    DCL CNTRL   INT2;
    DCL FACID   CHARACTER ( 3 );
    DCL ISINFO  INT4;
    DCL 01 FC2  FEEDBACK;

    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;
```

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 run-time option, or the heap was created with the KEEP attribute by CEECRHP.

Syntax



heap_id (input)
A fullword binary signed integer. *heap_id* is a token specifying the discarded heap.

A *heap_id* of 0 is invalid; 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Note

- 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.

For More Information

- For more information about the CEECRHP callable service, see "CEECRHP—Create New Additional Heap" on page 103.
- For more information about the CEEGTST callable service, and "CEEGTST—Get Heap Storage" on page 143.

Examples

- **C Example**

```
/*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);
   }
 /*  .
      .
      .  */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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 run-time 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.
```

- **PL/I Example**

```
*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  **/
```

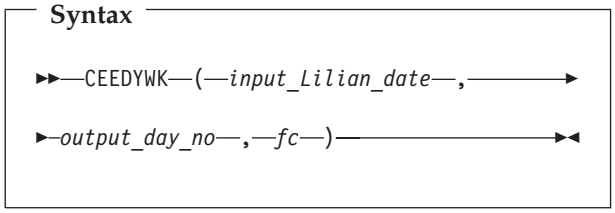
```
/** and then discard it.                          **/
/**                                               **/
/***************************************************/

PLIDSHP: PROC OPTIONS(MAIN);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL HEAPID INT4 ;
   DCL HPSIZE INT4 ;
   DCL INCR   INT4 ;
   DCL OPTS   INT4 ;
   DCL 01 FC  FEEDBACK ;

   DCL 01 FC2 FEEDBACK ;
   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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

Examples

- **C Example**

```
/*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;
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBMDYWK                        */
 /************************************************/
 /**                                              **/
 /** Function: CEEDYWK - calculate day of week    **/
 /** from Lilian date                             **/
 /**                                              **/
 /** In this example, CEEDYWK is called to        **/
 /** calculate the day of week on which a         **/
 /** Lilian date falls.                           **/
 /**                                              **/
 /************************************************/

 PLIDYWK: PROC OPTIONS(MAIN);
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL LILIAN  INT4 ;
    DCL DAYNUM  INT4 ;
    DCL 01 FC   FEEDBACK ;

    LILIAN = 152385; /* specify input as Lilian date */

    /* Call CEEDYWK to calculate the day of the      */
    /*    week on which Lilian date 152385 falls     */
    CALL CEEDYWK ( LILIAN , DAYNUM , FC );

    /* If CEEDYWK ran successfully, print result     */
    IF  FBCHECK( FC, CEE000)  THEN  DO;
       PUT SKIP LIST ( 'Lilian date ' || LILIAN
          || ' falls on a ' );
       SELECT (DAYNUM);
          WHEN (1) PUT LIST ( 'Sunday.' );
          WHEN (2) PUT LIST ( 'Monday.' );
          WHEN (3) PUT LIST ( 'Tuesday.' );
          WHEN (4) PUT LIST ( 'Wednesday.' );
          WHEN (5) PUT LIST ( 'Thursday.' );
          WHEN (6) PUT LIST ( 'Friday.' );
          WHEN (7) PUT LIST ( 'Saturday.' );
          END /* Case of DAYNUM */;
       END;
    ELSE  DO;
```

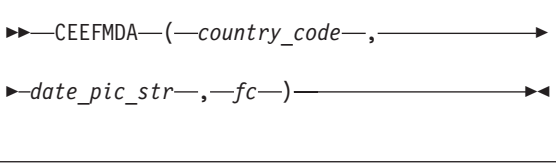
```
        DISPLAY( 'CEEDYWK failed with msg '
           || FC.MsgNo );
        STOP;
        END;


    END PLIDYWK;
```

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 run-time option or CEE5CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax



country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 227.

country_code is not case-sensitive. Also, if no value is specified, the default country code (as set by either the COUNTRY run-time option or the CEE5CTY callable service) is used.

If you specify an invalid *country_code*, the default date picture string 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

For More Information

- For a list of the default settings for a specified country, see Table 32 on page 227.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Library Reference*.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

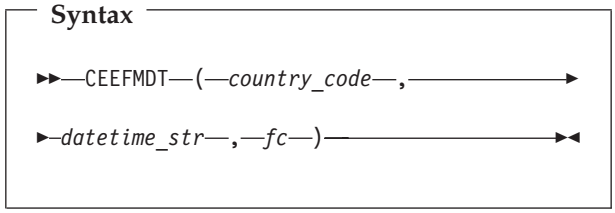
```
*PROCESS MACRO;
 /*Module/File name: IBMFMDA
 /************************************************/
 /**                                            **/
 /** Function: CEEFMDA - obtain default date    **/
 /**                     format                 **/
 /**                                            **/
 /************************************************/

 PLIFMDA: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;
    DCL COUNTRY CHARACTER ( 2 );
    DCL PICSTR  CHAR80;
    DCL 01 FC   FEEDBACK;

    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 run-time option or CEE5CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax



country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 227.

country_code is not case-sensitive. Also, if no value is specified, the default country code (as set by either the COUNTRY run-time option or by CEE5CTY) is used.

If you specify an invalid *country_code*, 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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-string* was returned.

For More Information

- For a list of the default settings for a specified country, see Table 32 on page 227.

- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of CEE5CTY, see "CEE5CTY—Set Default Country" on page 65.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Library Reference*.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

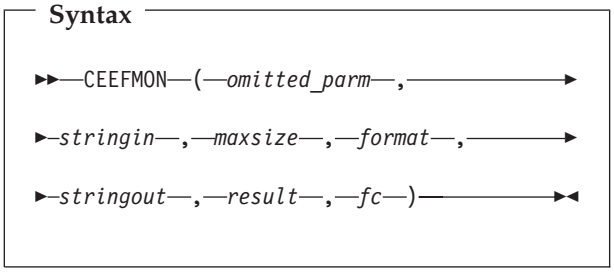
```
*PROCESS MACRO;
 /*Module/File name: IBMFMDT                 */
 /*******************************************/
 /**                                       **/
 /** Function: CEEFMDT - obtain default    **/
 /**                     date & time format **/
 /**                                       **/
 /*******************************************/

 PLIFMDT: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL COUNTRY CHARACTER ( 2 );
    DCL PICSTR  CHAR80;
    DCL 01 FC   FEEDBACK;

    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 LE/VSE settings from COUNTRY or CEE5CTY.

Syntax



omitted_parm
This parameter is reserved for future use. In C and PL/I, which allow you to omit parameters, it should be omitted. In COBOL, it should be coded as a dummy parameter. For more information about how to code an omitted parameter in C and PL/I, see "C Syntax" on page 58 and "PL/I Syntax" on page 59, respectively.

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

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

w The decimal digit string *w* specifies a minimum field width in which the result of the conversion is right-justified (or left-justified if *-w* is specified).

Left Precision

#n A # (pound sign) followed by the decimal digit string *n* specifies 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

.p A period followed by the decimal digit string *p* specifies 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

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)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

For More Information

- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()` and `strfmon()`, see *LE/VSE C Run-Time Library Reference*.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

  For C routines, the `strfmon()` function should be used instead of the CEEFMON service. For details, refer to the *LE/VSE C Run-Time Library Reference*.

- **COBOL Example**

```
 CBL LIB,APOST,RMODE(ANY)
*Module/File Name: IGZTFMON
************************************************
*  Example for callable service CEEFMON        *
*   Function: Convert a numeric value to a     *
*             monetary string using specified  *
*             format passed as parameter.      *
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBFMON.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  Monetary       COMP-2.
* OMIT is a dummy parameter used across LE call.
 01  OMIT           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  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 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 OMIT, 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.
```

- **PL/I Example**

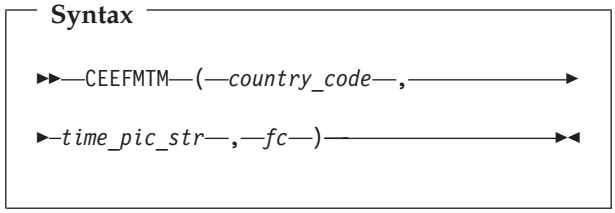
```
*PROCESS MACRO;
 /*Module/File Name: IBMFMON                          */
 /*************************************************/
 /* Example for callable service CEEFMON            */
 /* Function: Convert a numeric value to a monetary  */
 /*  string using specified format passed as parm    */
 /*************************************************/

 PLIFMON: PROC OPTIONS(MAIN);

 %INCLUDE CEEIBMAW; /* ENTRY defs, macro defs        */
 %INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants   */
 %INCLUDE CEEIBMLC; /* Locale category constants     */

 /* CEEFMON service call arguments */
 DCL MONETARY     FLOAT8;           /* input value   */
 DCL MAXSIZE_FMON INT4;             /* output size   */
 DCL FORMAT_FMON  CHAR(256) VARYING;/* format spec   */
 DCL RESULT_FMON  INT4;             /* result status */
 DCL OUTPUT_FMON  CHAR(60) VARYING; /* output string */

 DCL 01 FC        FEEDBACK;

   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 FC.Severity >  0 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 227.

CEEFMTM is affected only by the country code setting of the COUNTRY run-time option or CEE5CTY callable service, not the CEESETL callable service or the `setlocale()` function.

Syntax



country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 32 on page 227.

country_code is not case-sensitive. Also, if no value is specified, the default country code (as set by either the COUNTRY run-time option or the CEE5CTY callable service) is used.

If you specify an invalid *country_code*, 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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-string* was truncated and was not defined in CEEFMTM.

For More Information

- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.

- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Library Reference*.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*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  CHAR80;
    DCL 01 FC   FEEDBACK;

    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 LE/VSE 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 to free storage are conditional. If storage cannot be freed, 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.

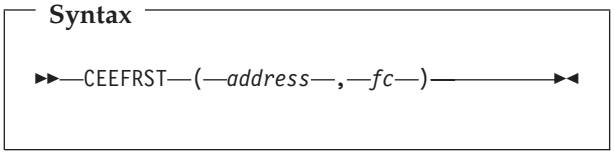
However, if you call CEEFRST for an invalid address, and you had specified TRAP(OFF), your application **can** abend. LE/VSE's reaction to this is undefined. Also, partial freeing of an allocated area is not supported.

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 run-time 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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 run-time 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.

For More Information

- For more information about the CEEGTST callable service, see
- "CEEGTST—Get Heap Storage" on page 143For further information about the HEAP run-time option, see "HEAP" on page 30.
- For more information about the CEECRHP callable service, see "CEECRHP—Create New Additional Heap" on page 103.
- For further information about the STORAGE run-time option, see "STORAGE" on page 42.

Examples

- **C Example**

```
/*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);
  }
/*  .
    .
    .  */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*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   INT4;
    DCL STGSIZE  INT4;
    DCL 01 FC    FEEDBACK;

    DCL 01 FC2   FEEDBACK;

    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;
```

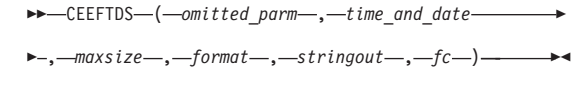
```
    END;
END PLIFRST;
```

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 in the time and date structure (TD_STRUCT) pointed to by the *time_and_date* parameter. 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 LE/VSE settings from COUNTRY or CEE5CTY.

Syntax



omitted_parm
This parameter is reserved for future use. In C and PL/I, which allow you to omit parameters, it should be omitted. In COBOL, it should be coded as a dummy parameter. For more information about how to code an omitted parameter in C and PL/I, see "C Syntax" on page 58 and "PL/I Syntax" on page 59, respectively.

time_and_date (input)
Points to the time structure that is to be converted.

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 11.

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.
- An optional decimal digit string 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.

- An optional precision value gives 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.
- The type of conversion is specified by one or two conversion characters. 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.

%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 59).

%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 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)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Note

- The values you specify in TD_STRUCT are not validated by CEEFTDS. If you specify any values outside the specified ranges, the results are unpredictable.

For More Information

- For more information about the CEESETL callable service and the locale LC_TIME category, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information about `setlocale()` and `strftime()`, see *LE/VSE C Run-Time Library Reference*.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

  For C routines, the `strftime()` function should be used instead of the CEEFTDS service. For details, refer to the *LE/VSE C Run-Time Library Reference*.

- **COBOL Example**

```
CBL LIB,APOST,RMODE(ANY)
*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.                       *
************************************************
 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.
*
* OMIT is a dummy parameter used across LE call.
 01  OMIT           COMP-2.
 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 OMIT, 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.
```

- **PL/I Example**

```
*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;

   /* 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

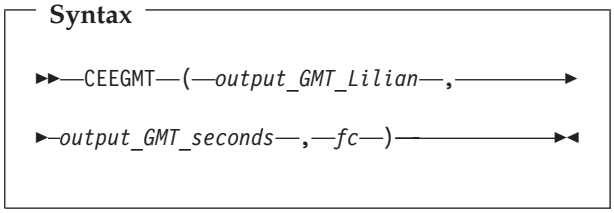
CEEGMT returns the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. GMT is also known as Coordinated Universal Time. The returned values are compatible with those generated and used by the other LE/VSE date and time services.

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.

CEEUTC is an alias of this service.

Syntax



output_GMT_Lilian (output)
A 32-bit binary integer representing the current date 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE2E6	3	2502	The UTC/GMT was not available from the system.
CEE3AU	1	3422	GMT was set to local time.

Usage Notes

- CEEDATE converts *output_GMT_Lilian* to a character date, and CEEDATM converts *output_GMT_seconds* to a character timestamp.
- **Caution:** If you use the DATE job control statement to override the system date, CEEGMT sets *output_GMT_Lilian* to the current local date, and *output_GMT_seconds* to the current local time, as returned by CEELOCT. CEEGMTO then returns GMT offset values of zero.

For More Information

- For more information about the CEEGMTO callable service, see "CEEGMTO—Get Offset from Greenwich Mean Time to Local Time" on page 137.
- For more information about the CEELOCT callable service, see "CEELOCT—Get Current Local Date or Time" on page 158.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

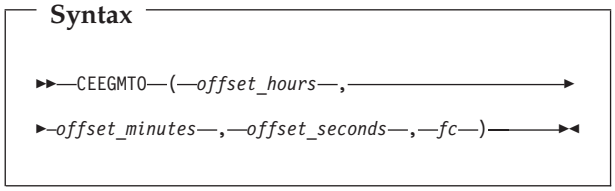
```
*PROCESS MACRO;
 /*Module/File name: IBMGMT                        */
 /************************************************/
 /**                                            **/
 /** Function: CEEGMT - get current Greenwich Mean **/
 /**           Time                             **/
 /** In this example, 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   INT4;
   DCL SECONDS  FLOAT8;
   DCL 01 FC    FEEDBACK;

   /* 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



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.

The range of *offset_hours* is −12 to +13 (+13 = Daylight Savings Time in the +12 time zone).

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). The range of *offset_seconds* is −43,200 to +46,800. *offset_seconds* can be used with CEEGMT to calculate local date and time. See "CEEGMT—Get Current Greenwich Mean Time" on page 135 for more information.

If the local time offset is not available from the system, *offset_seconds* is set to 0 and CEEGMTO 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.
CEE3AS	1	3420	The offset from UTC/GMT to local time was set to zero.

Usage Notes

- CEEDATM is used to convert number of seconds to a character timestamp.
- **Caution:** If you use the DATE job control statement to override the system date, CEEGMTO sets *offset_minutes* and *offset_seconds* to 0, and CEEGMT returns the current local time, as returned by CEELOCT.

For More Information

- For more information about the CEEDATM callable service, see "CEEDATM—Convert Seconds to Character Timestamp" on page 110.
- For more information about the CEELOCT callable service, see "CEELOCT—Get Current Local Date or Time" on page 158.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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, OR  '
                   SECONDS ' seconds. '
           ELSE
               DISPLAY 'CEEGMTO failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

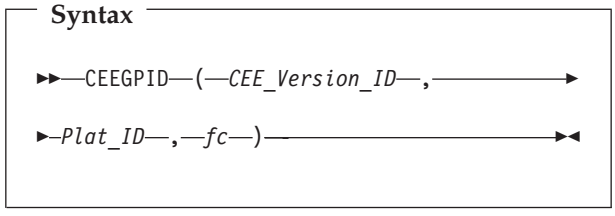
- **PL/I Example**

```
*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   INT4;
    DCL MINUTES INT4;
    DCL SECONDS FLOAT8;
    DCL 01 FC   FEEDBACK;

   /* 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 LE/VSE Version and Platform ID

CEEGPID retrieves the LE/VSE version ID and the platform ID of the version and platform of LE/VSE that is currently in use.

Syntax



CEE_Version_ID (output)

A fullword integer representing the version of LE/VSE that created this data block. The current value of this parameter is:

140 Version 1 Release 4 Modification level 0

141 Version 1 Release 4 Modification level 1

142 Version 1 Release 4 Modification level 2

143 Version 1 Release 4 Modification level 3

144 Version 1 Release 4 Modification level 4

Plat_ID (output)
A fullword integer representing the platform used for processing the current condition.

The current value of this parameter is:

5 VSE

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

Examples

- **C Example**

```
/*Module/File Name: EDCGPID   */

#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;

  /* 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);
   }
  printf("the LE version is %d",cee_ver_id);
  printf(" the current platform is ");
  switch (plat_id) {
     case 2: printf("OS/2\n");
             break;
     case 3: printf("OS390/VM/370\n");
             break;
     case 4: printf("AS/400\n");
             break;
     case 5: printf("VSE\n");
             break;
    default: printf("unrecognized platform id\n");
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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  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.

           DISPLAY 'Currently running version ' VERSION
                ' of IBM Language Environment'

      ** Evaluate PLATID to display this platform
           EVALUATE PLATID
             WHEN 2
               DISPLAY 'under OS/2'
             WHEN 3
               DISPLAY 'under OS390/VM/370'
             WHEN 4
               DISPLAY 'on an AS/400'
             WHEN 5
               DISPLAY 'under VSE'
           END-EVALUATE

           GOBACK.
```

- **PL/I Example**

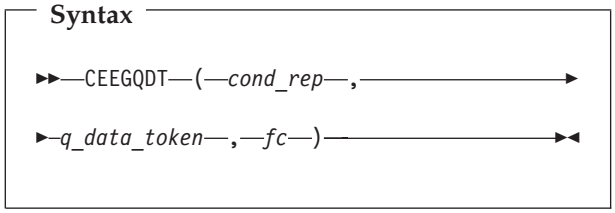
```
*PROCESS MACRO;
 /*Module/File name: IBMGPID                 */
 /*******************************************/
 /**                                       **/
 /** Function: CEEGPID - Get LE/VSE Version **/
 /** and Platform ID                       **/
 /**                                       **/
 /** 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  INT4;
    DCL PLATID   INT4;
    DCL 01 FC    FEEDBACK;

    /* 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 (5) PUT LIST( 'VSE');
            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



cond_rep (input)
A condition token defining the condition for which the *q_data_token* is retrieved.

q_data_token (output)
A 32-bit data object placed in the ISI by the CEESGL service.

fc (output)
An optional 12-byte condition token returned by CEEGQDT indicating the result of the service.

The following feedback codes can result from this service:

Symbolic Feedback 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.

For More Information

- For more information about the CEEHDLR callable service, see "CEEHDLR—Register User-Written Condition Handler" on page 146.
- For more information about the CEESGL callable service, see "CEESGL—Signal a Condition" on page 193.

Examples

- **C Example**

```
/*Module/File Name: EDCGQDT   */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);


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 */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

  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,APOST
      *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.

*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;
    DCL  TOKEN       BASED(@TOKEN)   INT4;
    DCL  RESULT      BASED(@RESULT)  INT4;
    DCL  01 NEWCOND  BASED(@NEWCOND) FEEDBACK;

    /* Local identifiers */
    DCL  QDATA       INT4;
    DCL  01 FC       FEEDBACK;

    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 with message '
          || 'number ' || CONDTOK.MsgNo);
       RESULT = 20 /* Percolate */;
       END;

    RETURN;

 END IBMGQDT;
```

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 AMODE(24), or if HEAP(,,BELOW) is in effect, the storage returned is always below the 16MB line. Above-the-line storage is returned only if the caller is in AMODE(31) 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 LE/VSE 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.

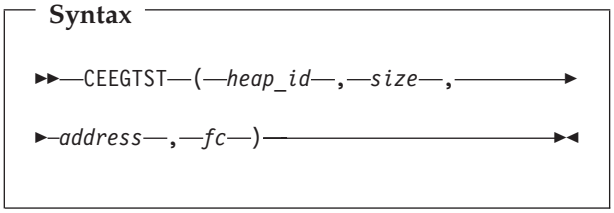
If you have specified a *heap_alloc_value* in the STORAGE run-time option, all storage allocated by CEEGTST is initialized to *heap_alloc_value*. Otherwise, it is left uninitialized.

If the value specified in the *size* parameter of CEEGTST is greater than the size of an increment (as specified in the HEAP run-time 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 run-time 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

run-time 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



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 machine address of the first byte of allocated storage. If storage cannot be obtained, *address* remains undefined. Storage is always allocated on a doubleword boundary.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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

- 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 LE/VSE callable storage services do not adjust the starting byte, you must be careful using callable services to allocate storage for PL/I structures. You must use fully defined structures and aggregates.
- CICS considerations—In a CICS environment, *size* should not exceed 65,504 bytes when HEAP(,,BELOW) is in effect or when running in AMODE(24), or 1024MB (1 gigabyte or X'40000000') when running in AMODE ANY and HEAP(,,ANY) is in effect. These CICS restrictions are subject to change from one release of CICS to another. Portable applications should respect current CICS limitations.

For More Information

- For more information about the HEAP run-time option, see "HEAP" on page 30.
- For more information about the CEESGL callable service, see "CEESGL—Signal a Condition" on page 193.
- For more information about the CEEFRST callable service, see "CEEFRST—Free Heap Storage" on page 129.
- For more information about the CEEDSHP callable service, see "CEEDSHP—Discard Heap" on page 119.

- For more information about the STORAGE run-time option, see "STORAGE" on page 42.
- For more information about the CEECRHP callable service, see "CEECRHP—Create New Additional Heap" on page 103.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 38.
- For information about how to change the size of storage obtained by CEEGTST, see "CEECZST—Reallocate (Change Size of) Storage" on page 105.

Examples

- **C Example**

```
/*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);
   }
  /* .
     .
     . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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                  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 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.
```

- **PL/I Example**

```
*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   INT4;
    DCL STGSIZE  INT4;
    DCL ADDRSS   POINTER;
    DCL 01 FC    FEEDBACK;

    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;
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:

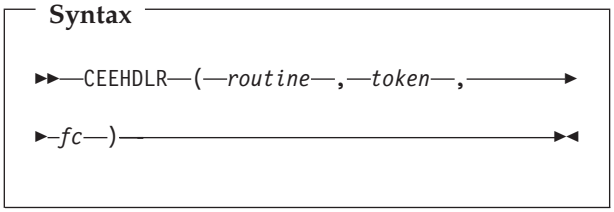
- It is registered for the current stack frame by CEEHDLR, and
- The LE/VSE condition manager requests the condition handler associated with the current stack frame to handle the condition.

LE/VSE places the user-written condition handlers associated with each stack frame in a queue. The queue can be empty at any given time. The LE/VSE 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 LE/VSE, but only when the associated stack frame is removed from the stack.

Note: PL/I cannot use CEEHDLR, because user-written condition handlers cannot be registered in PL/I.

Syntax



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Note

- COBOL consideration—You should not call CEEHDLR from a nested COBOL program.

For More Information

- For more information about the CEEHDLU callable service, see "CEEHDLU—Unregister User-Written Condition Handler" on page 149.

Examples

- **C Example**

```
/*Module/File Name: EDCHDLR   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);

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) {
 /* .
    .
    . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
      ***********************************************
      ** Return code will not be set after returning **
      ** from a condition signalled via CEESGL.      **
      ***********************************************
           MOVE ZERO TO RETURN-CODE.

           GOBACK.

CBL LIB,APOST,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.
```

- **Assembler Example**

```
CEEHDLRA TITLE 'Main program that registers a handler'
*
*        Symbolic Register Definitions and Usage
*
R1       EQU   1             Parm list addr, 0=no parms
R10      EQU   10            Base register for code
R12      EQU   12            LE 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,    Size of DSA          *
               MAIN=YES,     Program is a MAIN prog    *
               PPA=PPA1,     Our Program Prolog Area   *
               BASE=R10      Base register for code
         USING CEECAA,R12    Address for LE CAA
         USING CEEDSA,R13    Address for dynamic data
*
*        Announce ourselves
*
         WTO   'CEEHDLRA Says "HELLO"',ROUTCDE=11
*
*        Register User Handler
*
         LA    R1,USRHDLPP   Get addr of 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 plist
         CALL  CEEHDLR       Invoke CEEHDLR service
         CLC   FEEDBACK,=XL12'00' Check for success..
         BE    HDLRGOOD      Skip diags if success
*                            Failure.. issue diags
         WTO   '**** Call to CEEHDLR failed ****',     *
               ROUTCDE=11
         DUMP                Terminate prog 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 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 plist
         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)    Proc-ptr to Handlr
*
         LTORG ,                  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 Hdlr
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 Hdlr
HDLUPRM2 DS    A             Addr of Feedback Code
*
TOKEN    DS    F             32-bit Token: fullwd whose
*                                  *value* is 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 LE/VSE CAA
END   CEEHDLRA
```

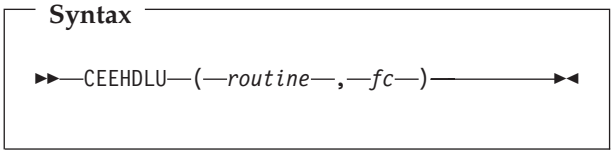
CEEHDLU—Unregister User-Written Condition Handler

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 LE/VSE, but only when the associated stack frame is removed from the stack.

Note: PL/I cannot use CEEHDLU, because user-written condition handlers cannot be registered in PL/I.

Syntax



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

For More Information

- For further information about specifying the *routine* parameter, see "CEEHDLR—Register User-Written Condition Handler" on page 146.

Examples

- **C Example**

```
/*Module/File Name: EDCHDLU   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);

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) {
 /* .
    .
    . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

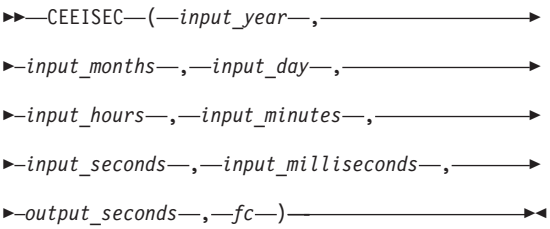
```
       WORKING-STORAGE SECTION.
       01  ROUTINE                 PROCEDURE-POINTER.
       01  TOKEN                   PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC 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.

CBL LIB,APOST,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



input_year (input)
A 32-bit binary integer representing the year.

The range of valid values for *input_year* is 1582 to 9999, inclusive.

input_month (input)
A 32-bit binary integer representing the month.

The range of valid values for *input_month* is 1 to 12.

input_day (input)
A 32-bit binary integer representing the day.

The range of valid values for *input_day* is 1 to 31.

input_hours (input)
A 32-bit binary integer representing the hours.

The range of valid values for *input_hours* is 0 to 23.

input_minutes (input)
A 32-bit binary integer representing the minutes.

The range of valid values for *input_minutes* is 0 to 59.

input_seconds (input)
A 32-bit binary integer representing the seconds.

The range of valid values for *input_seconds* is 0 to 59.

input_milliseconds (input)
A 32-bit binary integer representing milliseconds.

The range of valid values for *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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

Symbolic Feedback Code	Severity	Message Number	Message Text
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.

For More Information

- For more information about the CEESECS callable service, see "CEESECS—Convert Timestamp to Seconds" on page 188.
- For more information about the CEESECI callable service, see "CEESECI—Convert Seconds to Integers" on page 185.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

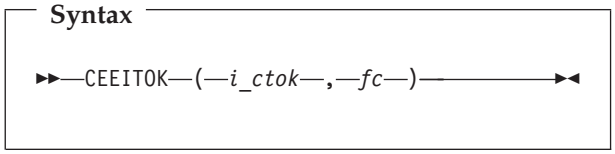
```
*PROCESS MACRO;
 /*Module/File name: IBMISEC                          */
 /*************************************************/
 /**                                             **/
 /** Function: CEEISEC - Convert integers to     **/
 /**                     seconds                 **/
 /**                                             **/
 /** 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     INT4;
    DCL MONTH    INT4;
    DCL DAYS     INT4;
    DCL HOURS    INT4;
    DCL MINUTES  INT4;
    DCL SECONDS  INT4;
```

```
   DCL MILLSEC  INT4;
   DCL OUTSECS  FLOAT8;
   DCL 01 FC    FEEDBACK;

   /* 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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

For More Information

- For information about how to decompose a condition token, see "CEEDCOD—Decompose a Condition Token" on page 117.

Examples

- **C Example**

```
/*Module/File Name: EDCITOK   */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);

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;
}
```

- **COBOL Example**

```
CBL LIB,APOST,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.
```

- **PL/I Example**

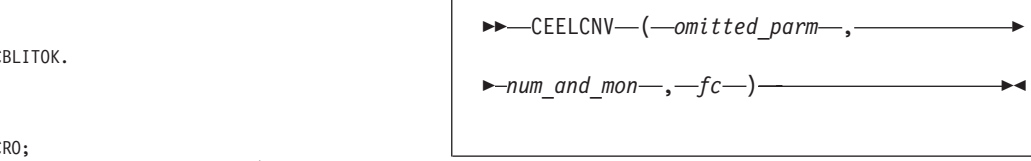
```
*PROCESS OPT(0), MACRO;
 /*Module/File name: IBMITOK                 */
 /*******************************************/
 /**                                       **/
 /** Function: CEEITOK - example of CEEITOK **/
 /**                     invoked from PL/I  **/
 /**                     ON-unit            **/
 /**                                       **/
 /*******************************************/

 IBMITOK:  PROCEDURE  OPTIONS(MAIN);

   %INCLUDE CEEIBMAW;
   %INCLUDE CEEIBMCT;

   DECLARE
      01 ITOKEN   FEEDBACK,
      01 FC       FEEDBACK,
      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, analogous to 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 LE/VSE settings from COUNTRY or CEE5CTY.

Syntax



omitted_parm
This parameter is reserved for future use. In C and PL/I, which allow you to omit parameters, it should be omitted. In COBOL, it should be coded as a dummy parameter. For more information about how to code an omitted parameter in C and PL/I, see "C Syntax" on page 58 and "PL/I Syntax" on page 59, respectively.

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_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 position 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.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.

Usage Notes

- 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

- For a description of LC_NUMERIC and LC_MONETARY categories, see "CEEQRYL—Query Active Locale Environment" on page 180.
- For details on the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()` and `localeconv()`, see *LE/VSE C Run-Time Library Reference*.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

  For C routines, the `localeconv()` function should be used instead of the CEELCNV service. For details, refer to the *LE/VSE C Run-Time Library Reference*.

- **COBOL Example**

```
CBL LIB,APOST,RMODE(ANY)
*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.   **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  MAINLCNV.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
************************************************
** Use Locale NM-Struct for CEELCNV calls      **
************************************************
 COPY CEEIGZNM.
*
 PROCEDURE DIVISION.
************************************************
**  Subroutine needed for addressing           **
************************************************
     CALL 'COBLCNV' USING NM-Struct.

     STOP RUN.
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBLCNV.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
* OMIT is a dummy parameter used across LE call.
 01  OMIT           COMP-2.
 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.
 LINKAGE SECTION.
************************************************
** Use Locale NM-Struct for CEELCNV calls      **
************************************************
 COPY CEEIGZNM.
*
 PROCEDURE DIVISION USING NM-Struct.
************************************************
```

```
      **  Call CEELCNV to retrieve values for locale**
      ************************************************
           CALL 'CEELCNV' USING OMIT,
                          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 4 TO LN-Length.
           MOVE 'FRAN' 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 **
      ************************************************
           IF Severity = 0
              CALL 'CEELCNV' USING OMIT,
                             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.
```

- **PL/I Example**

```
*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 CEEIBMNM; /* NM_STRUCT for CEELCNV calls  */

 /* use explicit pointer for local NM_STRUCT struct  */
 DCL NUM_AND_MON POINTER INIT(ADDR(NM_STRUCT));

 /* CEESETL service call arguments */
 DCL LOCALE_NAME CHAR(256) VARYING;

 DCL 01 FC       FEEDBACK;

   /* 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 = 'FRAN';

   /* 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 ( *, 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:

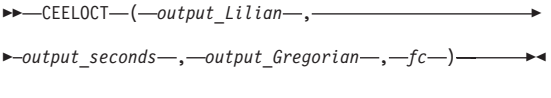
- Lilian date (the number of days since 14 October 1582)
- Lilian seconds (the number of seconds since 00:00:00 14 October 1582)
- Gregorian character string (in the form YYYYMMDDHHMISS999).

These values are compatible with other LE/VSE 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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3AT	1	3421	The local time was calculated using a date supplied in the JCL.
CEE2F3	3	2531	The local time was not available from the system.

Usage Notes

- **Caution:** If you use the DATE job control statement to override the system date, CEELOCT uses the date specified in the DATE statement to calculate the current local time. However, VSE does not increment the specified date at midnight. Therefore, if your job runs past midnight, the values CEELOCT returns in *output_Lilian*, *output_seconds*, and the date portion of *output_Gregorian* might not be what you expect.
- You can use the CEEGMT callable service to determine Greenwich Mean Time (GMT). Note, however, if you use the DATE job control statement to override the system date, CEEGMT sets GMT to the current local time, as returned by CEELOCT.
- You can use the CEEGMTO callable service to obtain the offset from GMT to local time. Note, however, if you use the DATE job control statement to override the system date, CEEGMTO sets the offset from GMT to local time to 0.

For More Information

- For more information about the CEEGMT callable service, see "CEEGMT—Get Current Greenwich Mean Time" on page 135.
- For more information about the CEEGMTO callable service, see "CEEGMTO—Get Offset from Greenwich Mean Time to Local Time" on page 137.
- For more information about the CEEDATM callable service, see "CEEDATM—Convert Seconds to Character Timestamp" on page 110.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

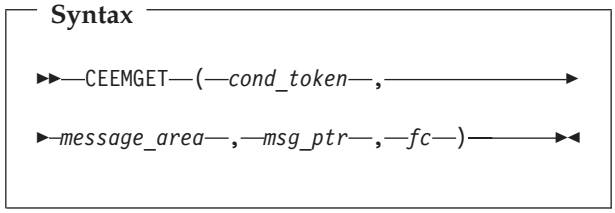
- **PL/I Example**

```
*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   INT4;
    DCL SECONDS  FLOAT8;
    DCL GREGORN  CHARACTER ( 17 );
    DCL 01 FC    FEEDBACK;

   /* 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 it or output it.

Syntax



cond_token (input)
A 12-byte condition token received as a feedback code from an LE/VSE 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* parameter should contain 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 the message has been truncated. When the entire message is returned, *msg_ptr* is zero.

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* parameter 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 *msg_ptr* contains 80. The next call does not add any blanks and starts with the 81st character.

fc (output)

A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

For More Information

- For a description of the 12-byte condition token constructed by the CEENCOD callable service, see "CEENCOD—Construct a Condition Token" on page 173.

Examples

- **C Example**

```
/*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);
  }
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

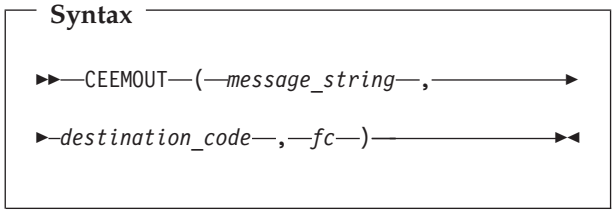
- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File Name: IBMMGET                          **/
 /*************************************************/
 /**                                              ***/
 /**Function       : CEEMGET - Get a Message       **/
 /**                                              ***/
 /*************************************************/
 PLIMGET: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL 01 CONTOK   FEEDBACK;
    DCL 01 FC       FEEDBACK;
    DCL MSGBUF      CHAR(80);
    DCL MSGPOINTER  INT4;

    /* 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



message_string (input)
A halfword prefixed string containing the message. DBCS characters must be enclosed within shift-out (byte X'0F') and shift-in (X'0E') characters.

Insert data cannot be placed in the message with CEEMOUT.

destination_code (input)
A 4-byte binary integer. The only accepted value for *destination_code* is 2. In the batch environment, LE/VSE writes the message to the *filename* of the file specified in the MSGFILE run-time option. Under CICS, the message is written to the CESE transient data queue.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 *filename* could not be located.

For More Information

- For more information about the MSGFILE run-time option, see "MSGFILE" on page 33.
- For more information on the CESE transient data queue, see *LE/VSE Programming Guide*.

Examples

- **C Example**

```
/*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);
  }
  /* .
     .
     . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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 run-time 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   **
      **     run-time 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.
```

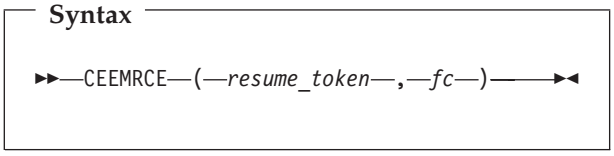
- **PL/I Example**

```
*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 filename  **/
 /** specified or defaulted in the MSGFILE run-time **/
 /** option.                                         **/
 /**                                                 **/
 /*****************************************************/
 PLIMOUT: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL MSGSTR VSTRING;
    DCL DESTIN INT4;
    DCL 01 FC FEEDBACK;

    MSGSTR = 'CEEMOUT ran successfully.';
                              /* Set message string */
    DESTIN = 2;  /* Send to MSGFILE filename         */
    /* 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 CEE5SRP. CEEMRCE is designed to be called from a user condition handler and works only in conjunction with the CEE5SRP service.

Syntax



resume_token (input)
An INT4 data type that contains a token, returned from the CEE5SRP service, representing the resume point in the user routine.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE07V	2	0255	The first argument passed to CEEMRCE was an unrecognized label.
CEE084	3	0260	No condition was active when a call to a condition management routine was made. The requested function was not performed.

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 “CEE5SRP—Set Resume Point” on page 94 for details on the CEE5SRP callable service.

COBOL Example of CEEMRCE Callable Service

```
CBL NODYNAM APOST
         IDENTIFICATION DIVISION.
         PROGRAM-ID. PGM2.
        *Module/File Name: IGZTMRCE
        ********************************************
        *                                          *
        * Sample program using CEE5SRP and CEEMRCE.*
        * PGM2 registers user-written condition    *
        * handler UCH1 using CEEHDLR.  It          *
        * sets a resume point using CEE5SRP. 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  CEEHDLR   PIC  X(08) VALUE 'CEEHDLR '.
             05  CEE5SRP   PIC  X(08) VALUE 'CEE5SRP '.
             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'.
             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 CEE5SRP USING RECOVERY-POINT,
                                    CASE-1
                 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.

         END-PGM2.
             GOBACK.

         END PROGRAM PGM2.

CBL NODYNAM APOST
         IDENTIFICATION DIVISION.
         PROGRAM-ID. PGM3.
        ********************************************
        *                                          *
        * Sample program using CEE5SRP 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  CEE5SRP   PIC  X(08) VALUE 'CEE5SRP '.
             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 CEE5SRP USING RECOVERY-POINT, FC
                 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.

         END-PGM3.
             GOBACK.

         END PROGRAM PGM3.

 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 CEE5SRP.                        *
*******************************************

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.
 END PROGRAM UCH1.
```

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.

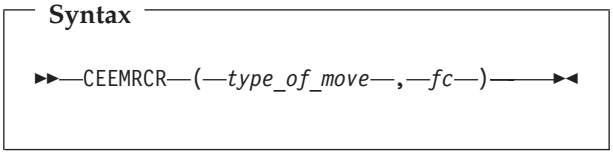
Initially, the resume cursor is placed after the machine instruction that caused the condition. Whenever CEEMRCR moves the resume cursor and passes stack frames, associated exit routines are invoked. Note that "exit routine" refers to user condition handlers as well as language-specific condition handlers. This moving also unregisters any associated user condition handlers. 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 LE/VSE 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



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.

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. The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE07U	1	0254	The first argument 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.

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 LE/VSE 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.

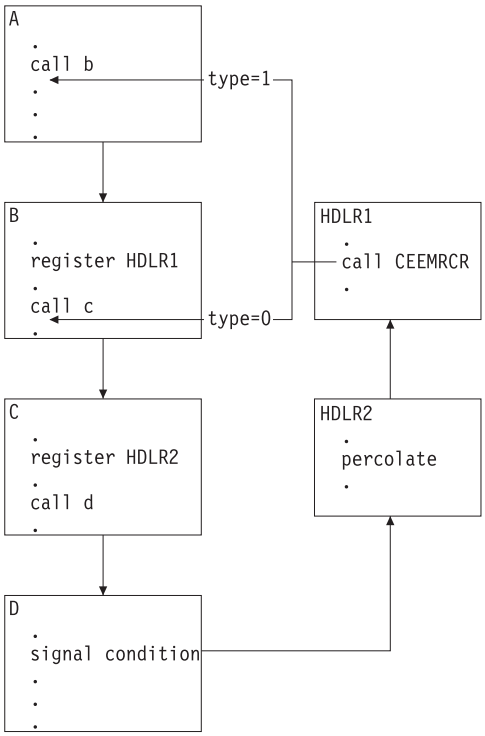


Figure 9. Moving Resume Cursor Using CEEMRCR

The same scenario is illustrated in Figure 10 on page 168, 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.

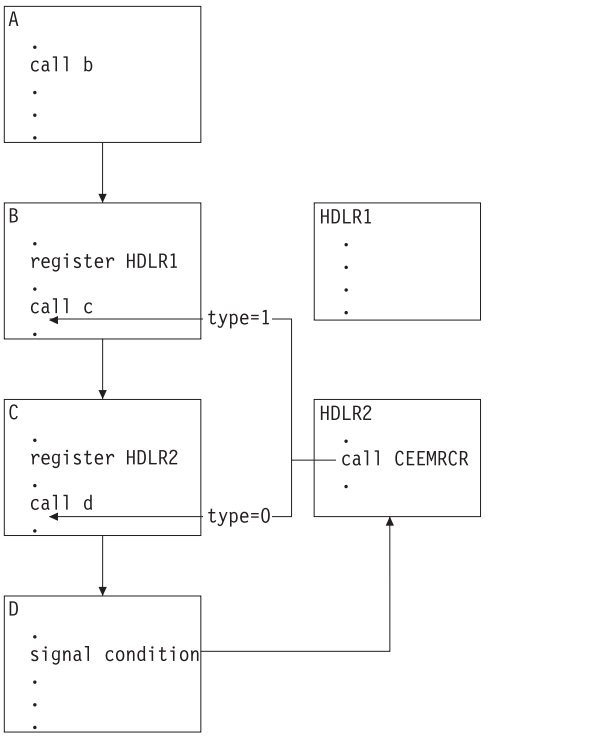


Figure 10. Moving Resume Cursor Using CEEMRCR

In Figure 11, 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.

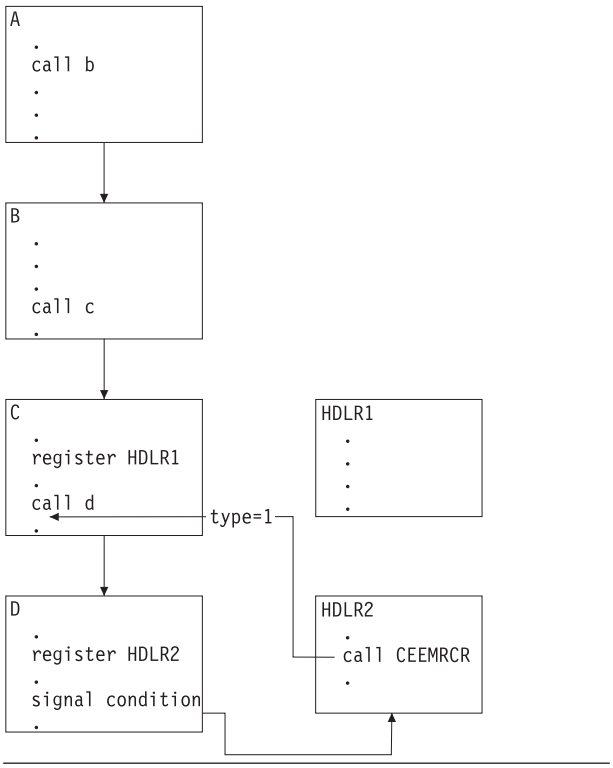


Figure 11. Moving Resume Cursor Using CEEMRCR

Examples

- **C Example**

```
/*Module/File Name: EDCMRCR   */

#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);

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;
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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'
      ************************************************
      ** Return code will not be set after returning **
      ** from a condition signalled via CEESGL.      **
      ************************************************
           MOVE ZERO TO RETURN-CODE.
           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.
```

- **PL/I Example**

  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,APOST,NOOPT
      *Module/File Name: IGZTMRCP
      *************************************************
      **                                             **
      ** IGZTMRCP - Drive sample program for CEEMRCR.**
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTMRCP.

       PROCEDURE DIVISION.

       INVOKE-DRIVER.
           CALL 'RUNMRCR'.
           DISPLAY '==> Execution resumed in routine '
               'which called the routine '
               'which registered handler'.
           STOP RUN.

       END PROGRAM IGZTMRCP.

CBL LIB,APOST,NOOPT
      *************************************************
      **                                             **
      ** RUNMRCR - Run sample program for CEEMRCR.   **
      **                                             **
      *************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. RUNMRCR.
       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 'HDLMRCR'.
     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.
************************************************
*    (exception occurs and is handled)         **
************************************************
     DISPLAY '==> Execution resumed in routine '
         'which registered handler'.

 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
     END-IF.

     STOP RUN.
 END PROGRAM RUNMRCR.

************************************************
**                                            **
** HDLMRCR - Invoke PL/I program to handle    **
**                     current condition.     **
**                                            **
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. HDLMRCR.
 DATA DIVISION.
 LINKAGE SECTION.
 01  TOKEN                   PIC S9(9) BINARY.
 01  RESULT                  PIC S9(9) BINARY.
 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.

 INVOKE-PLI.
     CALL 'IBMMRCR' USING ADDRESS OF CURCOND,
                          ADDRESS OF TOKEN,
                          ADDRESS OF RESULT,
                          ADDRESS OF NEWCOND.
          GOBACK.

      END PROGRAM HDLMRCR.
```

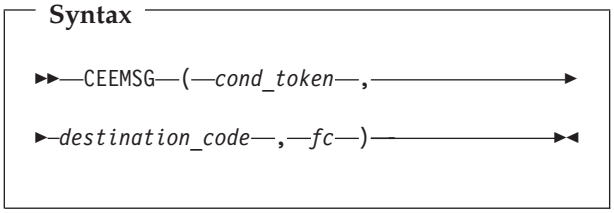
```
*PROCESS OPT(0), MACRO,SOURCE,INSOURCE;
 /* Module/File name: IBMMRCR                      */
 /************************************************/
 /**                                              */
 /** Function: CEEMRCR - move resume cursor relative */
 /**                     to handle cursor         */
 /**                                              */
 /************************************************/
 IBMMRCR:  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;
    DCL  TOKEN       BASED( @TOKEN )   INT4;
    DCL  RESULT      BASED( @RESULT )  INT4;
    DCL  01 NEWCOND  BASED( @NEWCOND ) FEEDBACK;

    /* Local identifiers */
    DCL MOVETYP INT4;
    DCL 01 FC FEEDBACK;

    IF  FBCHECK( CONDTOK, CEE349 )  THEN  DO;
       /* Expeced fixed-point divide exception occurred */

       MOVETYP = 1 /* Move resume cursor to stack frame */
                   /* of program which called routine   */
                   /* which registered this handler    */;
       CALL CEEMRCR ( MOVETYP, FC );
       IF  FBCHECK( FC, CEE000)  THEN  DO;
          PUT SKIP LIST( 'Resume cursor moved one stack'
             || ' prior to position of Handler Cursor,' );
          PUT SKIP LIST( ' and Handler Cursor moved to '
             || 'first handler of that stack frame' );
          RESULT = 10 /* Resume */;
          END;
       ELSE  DO;
          DISPLAY( 'CEEMRCR failed with msg '
             || FC.MsgNo );
          RESULT = 30 /* Promote */;
          NEWCOND = FC;
          END;

       END;
    ELSE /* Unexpected condition -- percolate */  DO;
       DISPLAY( 'User condition handler entered for '
          || CONDTOK.FacID || ' condition with message '
          || 'number ' || CONDTOK.MsgNo );
       RESULT = 20 /* Percolate */;
       DISPLAY('Percolating...');
       END;

    RETURN;

 END IBMMRCR;
```

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 LE/VSE service that returns a condition token.

Syntax



cond_token (input)
A 12-byte condition token received as the result of an LE/VSE callable service.

destination_code (input)
A 4-byte binary integer. The only valid value for *destination_code* is 2. In the batch environment, LE/VSE writes the message to the *filename* of the file specified in the MSGFILE run-time option. Under CICS, the message is written to the CESE transient data queue.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 *filename* 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.

For More Information

- For more information about the MSGFILE run-time option, see "MSGFILE" on page 33.
- For more information about the CESE transient data queue, see *LE/VSE Programming Guide*.

Examples

- **C Example**

```
/*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);
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZTMSG
      ************************************************
      **                                            **
      ** CBLMSG - Call CEEMSG to get, format and    **
      **                    dispatch a message      **
      **                                            **
      ** In this example, CEE5MDS is called with an **
      ** invalid country code so that a condition   **
      ** token would be returned to use as input to **
      ** Any LE service could have been called.     **
      ** CEEMSG uses the condition token to get,    **
      ** format and dispatch the message associated **
      ** with the condition that occurred in CEE5MDS.**
      **                                            **
      ************************************************
       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 any LE service, CEE5MDS 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 'CEE5MDS' USING COUNTRY, DECSEP, FC.
        PARA-CBLMSG.
      ************************************************
      ** Specify 2 for destination, so message will  **
      ** be written to the ddname specified or       **
      ** defaulted in the MSGFILE run-time option.   **
      ************************************************
           MOVE 2 TO MSGDEST.
      ************************************************
      ** Call CEEMSG using the FC returned from      **
      **     CEE5MDS 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.
```

- **PL/I Example**

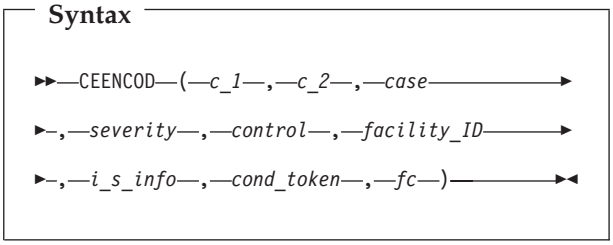
```
*PROCESS LANGLVL(SAA), MACRO;
 /*Module/File name: IBMMSG                          */
 /*****************************************************/
 /**                                                 **/
 /** Function: CEEMSG  - get, format and dispatch    **/
 /**                     a message                   **/
 /**                                                 **/
 /** In this example, CEE5MDS is called with an      **/
 /** invalid country code so that a condition token  **/
 /** would be returned to use as input to CEEMSG.    **/
 /** Any LE/VSE could have been called. CEEMSG uses  **/
 /** the condition to get, format and dispatch the   **/
 /** message associated with the condition that      **/
 /** occurred in CEE5MDS                             **/
 /**                                                 **/
 /*****************************************************/
 PLIMSG: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL COUNTRY  CHARACTER ( 2 );
    DCL DECSEP   CHARACTER ( 2 );
    DCL 01 FC    FEEDBACK;
    DCL MSGDEST  INT4;
    DCL 01 FC2   FEEDBACK;

    COUNTRY = 'LN';  /* Specify an invalid country   */
                     /* code to receive a non-zero   */
                     /* feedback code                */

    /* Call any service (CEE5MDS in this case) to    */
    /* receive a condition token that CEEMSG will    */
    /* format and dispatch a message                 */
    CALL CEE5MDS ( COUNTRY, DECSEP, FC );

    MSGDEST = 2; /* Specify 2 as destination, so     */
                 /* msg will go to filename speci-   */
                 /* fied in MSGFILE run-time 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 LE/VSE.

The condition token communicates with the LE/VSE message and condition handling callable services, and user routines. Also, all LE/VSE callable services use the condition-token data type to return information to the user as a feedback code.

Syntax



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—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 an LE/VSE 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 run-time 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 *control* (input) parameter, above, 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 LE/VSE condition manager, insert data is generated describing the particular instance of its occurrence is generated. This insert data is used, for example, to write to a file a message associated with that particular 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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

- The condition_ID is a fullword identifier that, with the *facility_ID* parameter of CEENCOD, describes the condition that the token communicates. The condition_ID contains two case fields; these differ depending on whether they are case 1 or case 2 types. These cases represent the types of conditions that the condition token will communicate.

  Case 1 - service condition
  Case 1 is used by all LE/VSE callable services and most applications. It contains:

  severity
  a 2-byte binary integer specifying the severity of the condition. You can specify the severity in the *c_1* parameter of CEENCOD.

  msg_no
  a 2-byte binary number that identifies the message associated with the condition. You can specify the msg_no in the *c_2* parameter of CEENCOD.

  Case 2 - class/cause code condition
  Case 2 is used by some operating systems and compiler run-time libraries to modify a particular message or condition. It contains:

  class_code
  a 2-byte, binary number that identifies the message id (same type as **severity** above) associated with the **class** of the condition. You can specify class_code in the *c_1* parameter of CEENCOD.

  cause_code
  a 2-byte, binary number that identifies the message subid (same type as msg_no, above) associated with the **cause** of the condition. You can specify cause_code in the *c_2* parameter of CEENCOD.

  Note: The class_code and cause_code fields are assigned by products other than LE/VSE.

- C considerations—The structure of the condition token (type_FEEDBACK) is described in the `leawi.h` header file shipped with LE/VSE. You 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:

```
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

For More Information

- For more information about condition tokens, see *LE/VSE Programming Guide*.

Examples

- **C Example**

```
/*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);
  }
 /* .
    .
    . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

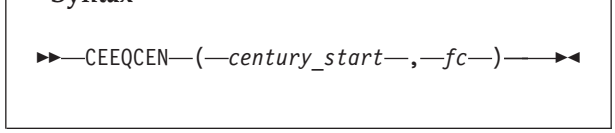
- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBMNCOD                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEENCOD - construct a condition token */
 /**                                                  */
 /*****************************************************/
 PLINCOD: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL SEV        INT2;
    DCL MSGNO      INT2;
    DCL CASE       INT2;
    DCL SEV2       INT2;
    DCL CNTRL      INT2;
    DCL FACID      CHARACTER ( 3 );
    DCL ISINFO     INT4;
    DCL 01 NEWTOK  FEEDBACK;
    DCL 01 FC      FEEDBACK;

    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. */

    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 LE/VSE 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



century_start (output)
An integer between 0 and 100 indicating the year on which the century window is based.

For example, if the LE/VSE 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 LE/VSE that, in 1996, all 2-digit years lie within the 100-year window starting 80 years before the system date (between 1916 and 2015, inclusive).

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

For More Information

- For more information about the CEESCEN callable service, see "CEESCEN—Set the Century Window" on page 182.

Examples

- **C Example**

```
/*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);
     }
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZTQCEN
      ************************************************
      **                                            **
      ** CBLQCEN - Call CEEQCEN to query the LE     **
      **           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   **
      ** windoe within which Language Environment    **
      ** 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.
```

- **PL/I Example**

```
*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 INT4;
    DCL 01 FC   FEEDBACK;

    /* 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 '
```

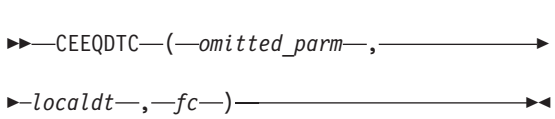
```
          || FC.MsgNo );
        STOP;
        END;

    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 LE/VSE settings from COUNTRY or CEE5CTY.

Syntax



omitted_parm
This parameter is reserved for future use. In C and PL/I, which allow you to omit parameters, it should be omitted. In COBOL, it should be coded as a dummy parameter. For more information about how to code an omitted parameter in C and PL/I, see "C Syntax" on page 58 and "PL/I Syntax" on page 59, respectively.

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:

Keyword	Meaning
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)

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.

Usage Note

- If no call to CEESETL has been made, the default locale values to be used are determined at installation time for LE/VSE.

For More Information

- For more information about the current locale, and the CEESETL callable service, see

"CEESETL—Set Locale Operating Environment" on page 191.

- For more information on `setlocale()` and `localdtconv()`, see *LE/VSE C Run-Time Library Reference*.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

  For C routines, the `localdtconv()` function should be used instead of the CEEQDTC service. For details, refer to the *LE/VSE C Run-Time Library Reference*.

- **COBOL Example**

```
CBL LIB,APOST,RMODE(ANY)
*Module/File Name: IGZTQDTC
************************************************
*  Example for callable service CEEQDTC        *
*  MAINQDTC - Retrieve date and time convention *
*             structures for two countries and  *
*             compare an item.                  *
************************************************
 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.
* OMIT is a dummy parameter used across LE call.
 01  OMIT           COMP-2.
 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 OMIT,
                          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 OMIT,
                          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.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File Name: IBMQDTC                          */
 /*************************************************/
 /* Example for callable service CEEQDTC            */
 /* Function: Retrieve date and time convention     */
 /*  structures for two countries, compare an item. */
```

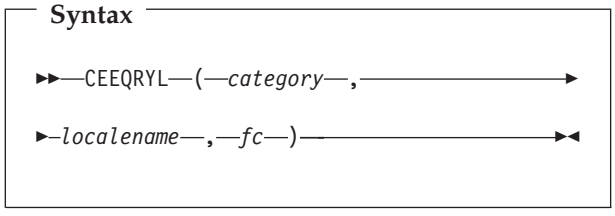
```
/*************************************************/

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;

  /* 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 LE/VSE settings from COUNTRY or CEE5CTY.

Syntax



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.

LE/VSE locale callable services do not support the LC_TOD and LC_SYNTAX categories.

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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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

- CEEQRYL does not change the status of the active locales.
- If the active locale is not explicitly set with CEESETL or `setlocale(category, localename)`, the `SAA C` locale is chosen, and querying the locale with CEEQRYL returns "C" as the locale name.

For More Information

- For more information about the current locale, and the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Library Reference*.
- For more information about LC_TIME, see "CEEQDTC—Query Locale Date and Time Conventions" on page 178.
- For information on the definition of the `SAA C` locale, see *LE/VSE C Run-Time Programming Guide*.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

For examples of how to use CEEQRYL in combination with other LE/VSE locale callable services, see "CEESETL—Set Locale Operating Environment" on page 191.

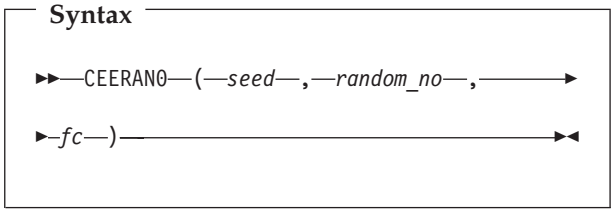
CEERAN0—Calculate Uniform Random Numbers

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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE2ES	3	2524	An invalid seed value was passed to CEERAN0. The random number was set to -1.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBMRAN0                          */
 /*************************************************/
 /**                                              **/
 /** Function: CEERAN0 - calculate uniform random   **/
 /** numbers                                      **/
 /**                                              **/
 /*************************************************/
 PLIRAN0: PROC OPTIONS(MAIN);
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL SEED     INT4;
    DCL RANDNUM  FLOAT8;
    DCL 01 FC    FEEDBACK;

    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;
```

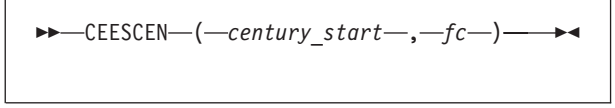
CEESCEN—Set the Century Window

CEESCEN sets the century in which LE/VSE 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 LE/VSE default century interval does not meet the requirements of a particular application.

To query the century window, use CEEQCEN.

Syntax



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 1996, therefore, all two-digit years are assumed to represent dates between 1916 and 2015, inclusive.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE2E6	3	2502	The UTC/GMT was not available from the system.
CEE2F5	3	2533	The value passed to CEESCEN was not between 0 and 100.

For More Information

- For more information about the CEEDAYS callable service, see "CEEDAYS—Convert Date to Lilian Format" on page 113.
- For more information about the CEESECS callable service, see "CEESECS—Convert Timestamp to Seconds" on page 188.
- For more information about the CEEQCEN callable service, see "CEEQCEN—Query the Century Window" on page 176.

Examples

- **C Example**

```
/*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);
  }
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZTSCEN
      ************************************************
      **                                            **
      ** CBLSCEN - Call CEESCEN to set the LE       **
      **           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.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBMSCEN                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEESCEN - set the century window       */
 /**                                                  */
 /*****************************************************/
 PLISCEN: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL STARTCW  INT4;
    DCL 01 FC    FEEDBACK;
    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;
```

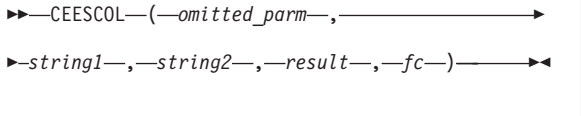
CEESCOL—Compare Collation Weight of Two Strings

CEESCOL, 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 LE/VSE settings from COUNTRY or CEE5CTY.

Syntax



omitted_parm
This parameter is reserved for future use. In C and PL/I, which allow you to omit parameters, it should be omitted. In COBOL, it should be coded as a dummy parameter. For more information about how to code an omitted parameter in C and PL/I, see "C Syntax" on page 58 and "PL/I Syntax" on page 59, respectively.

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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE3T1	3	4001	General Failure: Service could not be completed.

For More Information

- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()` and `strcoll()`, see *LE/VSE C Run-Time Library Reference*.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **COBOL Example**

```
CBL LIB,APOST,RMODE(ANY)
*Module/File Name: IGZTSCOL
************************************************
*  Example for callable service CEESCOL        *
*   COBSCOL - Compare two character strings     *
*             and print the result.             *
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBSCOL.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
* OMIT is a dummy parameter used across LE call.
 01  OMIT           COMP-2.
 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 OMIT, 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.
```

- **PL/I Example**

```
*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 LE */
 %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;

   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;
```

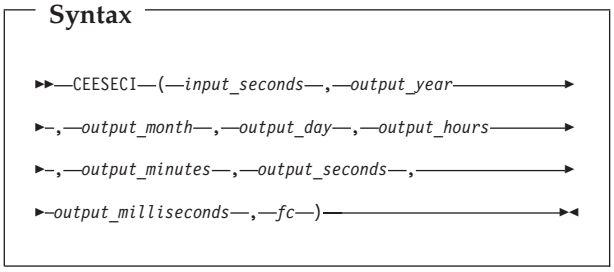
CEESECI—Convert Seconds to Integers

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 separate binary integers representing year, month, day, hour, second, and millisecond to a 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



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 range of valid values 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 values for *output_year* is 1582 to 9999, inclusive.

output_month (output)
A 32-bit binary integer representing the month.

The range of valid values for *output_month* is 1 to 12.

output_day (output)
A 32-bit binary integer representing the day.

The range of valid values for *output_day* is 1 to 31.

output_hours (output)
A 32-bit binary integer representing the hour.

The range of valid values for *output_hours* is 0 to 23.

output_minutes (output)
A 32-bit binary integer representing the minutes.

The range of valid values for *output_minutes* is 0 to 59.

output_seconds (output)
A 32-bit binary integer representing the seconds.

The range of valid values for *output_seconds* is 0 to 59.

output_milliseconds (output)
A 32-bit binary integer representing milliseconds.

The range of valid values for *output_milliseconds* is 0 to 999.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

For More Information

- For more information about the CEEISEC callable service, see "CEEISEC—Convert Integers to Seconds" on page 151.

Examples

- **C Example**

```
/*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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBMSECI                          */
 /*************************************************/
 /**                                                 */
 /** Function: CEESECI - convert seconds to integers */
 /**                                                 */
 /*************************************************/
 PLISECI: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

    DCL INSECS   FLOAT8;
    DCL YEAR     INT4;
    DCL MONTH    INT4;
    DCL DAYS     INT4;
    DCL HOURS    INT4;
    DCL MINUTES  INT4;
    DCL SECONDS  INT4;
    DCL MILLSEC  INT4;
    DCL 01 FC    FEEDBACK;

    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

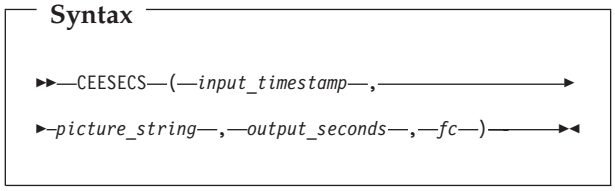
CEESECS converts a string representing a timestamp into the number of Lilian seconds (number of seconds since 00:00:00 14 October 1582). This service makes it easier to perform time arithmetic, such as calculating the elapsed time between two timestamps.

CEESECS is affected only by the country code setting of the COUNTRY run-time option or CEE5CTY 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 1996, all 2-digit years represent dates between 1916 and 2015, inclusive. You can change this range by using the callable service CEESCEN.

Syntax



input_timestamp (input)
A halfword length-prefixed character string (VSTRING), 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.

If any part or all of the time value is omitted, zeros are substituted for the remaining values. For example:

```
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 run-time option For example, if the current value of the COUNTRY run-time 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 234 for a list of Japanese eras supported by CEESECS.

If *picture_string* includes a Chinese Era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in *input_timestamp* represents the year number within the Chinese Era. For example, the year 1988 equals the Chinese year 77 in the MinKow Era.

See Table 36 on page 235 for a list of Chinese Eras supported by CEESECS.

See Table 33 on page 233 for a list of valid picture characters, and Table 34 on page 234 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 Japanese or Chinese 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 Japanese (<JJJJ>) or Chinese (<CCCC>) 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 Note

- 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.

For More Information

- For more information about the CEESCEN callable service, see "CEESCEN—Set the Century Window" on page 182.
- For more information about the COUNTRY run-time option, see "COUNTRY" on page 26.
- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()`, see *LE/VSE C Run-Time Library Reference*.

Examples

- **C Example**

```
/*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,
            "Mmmmmmmmmmz 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);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*PROCESS MACRO;
 /*Module/File name: IBMSECS                          */
 /*****************************************************/
 /**                                                  */
 /** Function: CEESECS - Convert timestamp to seconds */
 /**                                                  */
 /** In this example, CEESECS is called to return an  */
```

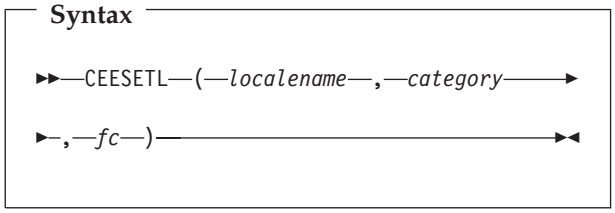
```
/** input timestamp as the number of seconds since   */
/** 14 October 1582.                                 */
/**                                                  */
/****************************************************/
PLISECS: PROC OPTIONS(MAIN);

   %INCLUDE  CEEIBMAW;
   %INCLUDE  CEEIBMCT;

   DCL TIMESTP VSTRING;
   DCL PICSTR  VSTRING;
   DCL SECONDS FLOAT8;
   DCL 01 FC   FEEDBACK;

   TIMESTP = '10 November 1992'; /* Specify input   */
                                 /* date as timestamp */
   PICSTR  = 'ZD Mmmmmmmmmmmmmz 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 affects only locales and those callable services and language functions that use them. CEESETL does not affect the settings controlled by the COUNTRY run-time option or the CEE5CTY callable service.

Syntax



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 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.

LE/VSE 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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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

- 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,
- If the active locale is not explicitly set with CEESETL or `setlocale(category, localename)`, then the `SAA C` locale is chosen, and querying the locale with CEEQRYL returns "C" as the locale name.

For More Information

- For more information on `setlocale()`, see *LE/VSE C Run-Time Library Reference*.
- For details of how various locale categories affect C language functions, see *LE/VSE C Run-Time Library Reference*
- For more information about specifying environment variables to set the locale, see *LE/VSE C Run-Time Programming Guide*.
- For information on the definition of the `SAA C` locale, see *LE/VSE C Run-Time Programming Guide*.
- For more information about LC_TIME, see "CEEQDTC—Query Locale Date and Time Conventions" on page 178.
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

  For C routines, the `setlocale()` function should be used instead of the CEESETL service. For details, refer to the *LE/VSE C Run-Time Library Reference*.

- **COBOL Example**

```
CBL LIB,APOST,RMODE(ANY)
*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.
```

- **PL/I Example**

```
*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;

   /* 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 LE/VSE 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 LE/VSE condition manager to identify and react to conditions.

CEESGL is typically used to generate application-specific conditions that are recognized by condition handlers registered via CEEHDLR. Conditions can also be selected to simulate an LE/VSE 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 an LE/VSE condition.

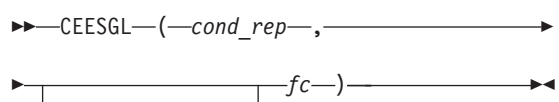
Unique conditions signaled by CEESGL are considered to be enabled under LE/VSE. Therefore, they undergo LE/VSE condition handling.

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 run-time option), 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.

Table 28 on page 92 contains a list of the S/370 program interrupt codes and their corresponding LE/VSE condition token names and message numbers.

Syntax



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 LE/VSE has already defined.

Conditions signaled by CEESGL are not necessarily handled by LE/VSE. If you call CEESGL with a *cond_rep*, LE/VSE passes control to the language in which the routine is written. The language condition handler then determines whether it should handle the condition. If so, the HLL handles the condition. If not, control returns to the LE/VSE condition manager and LE/VSE condition handling continues. 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.

*q_data token*s associated with a condition using CEESGL can be extracted later using the CEEGQDT callable service.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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

- After the return from a call to CEESGL, the return code might contain an unpredictable value, and might need to be reset. See the COBOL sample program in "Examples" on page 146 for an example.
- PL/I consideration—Conditions with a facility_ID of IBM cannot be used in CEESGL.

For More Information

- For more information about the CEECMI callable service, see "CEECMI—Store and Load Message Insert Data" on page 101.
- For more information about the ERRCOUNT run-time option, see "ERRCOUNT" on page 29.
- To construct your own condition token, see "CEENCOD—Construct a Condition Token" on page 173.
- For more information about condition handling, see *LE/VSE Programming Guide*.
- For more information about the CEEHDLR callable service, see "CEEHDLR—Register User-Written Condition Handler" on page 146.
- For more information about the CEEGQDT callable service, see "CEEGQDT—Retrieve q_data_token" on page 140.

Examples

- **C Example**

```
/*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 &&
       _FBCHECK ( fc , CEE069 ) != 0 &&
       _FBCHECK ( fc , CEE0CE ) != 0 &&
       _FBCHECK ( fc , CEE0CF ) != 0 ) {
     printf("CEENCOD failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }

  /* signal the condition */
  CEESGL(&condtok,&qdata,&fc);
  if ( _FBCHECK ( fc , CEE069 ) != 0 ) {
     printf("CEESGL failed with message number %d\n",
            fc.tok_msgno);
     exit(2999);
  }
 /* .
    .
    . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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 AND
              NOT CEE069 of FC AND
              NOT CEE0CE of FC AND
              NOT CEE0CF of FC
               DISPLAY 'CEESGL  failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

- **PL/I Example**

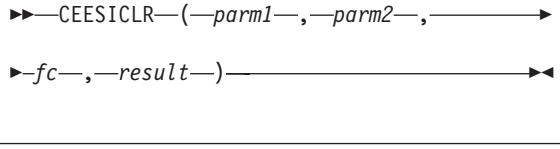
```
*PROCESS MACRO;
 /*Module/File name: IBMSGL                 */
 /*******************************************/
 /**                                        */
 /** Function: CEESGL  - Signal a Condition */



 /**                                        */
 /*******************************************/
 PLISGL: PROC OPTIONS(MAIN);

    %INCLUDE  CEEIBMAW;



    %INCLUDE  CEEIBMCT;
    DCL 01 CONDTOK  FEEDBACK;
    DCL QDATA       INT4;
    DCL 01 FC       FEEDBACK;



    /* 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)  |
        FBCHECK( FC, CEE069)  |
        FBCHECK( FC, CEE0CE)  |
        FBCHECK( FC, CEE0CF)  THEN  DO;
```

```
      PUT SKIP LIST( 'Condition CEE001 signalled' );
      END;
   ELSE  DO;
      DISPLAY( 'CEESGL  failed with msg '
         || FC.MsgNo );



      STOP;
      END;

END PLISGL;

  *PROCESS MACRO;
   /*Module/File name: IBMSGL                  */
   /*********************************************/
   /**                                         */
   /** Function: CEESGL  - Signal a Condition  */
   /**                                         */
   /*********************************************/
   PLISGL: PROC OPTIONS(MAIN);

      %INCLUDE  CEEIBMAW;
      %INCLUDE  CEEIBMCT;
      DCL 01 CONDTOK  FEEDBACK;
      DCL QDATA       INT4;
      DCL 01 FC       FEEDBACK;
      /* 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)  |
          FBCHECK( FC, CEE069)  |
          FBCHECK( FC, CEE0CE)  |
          FBCHECK( FC, CEE0CF)  THEN  DO;
        PUT SKIP LIST( 'Condition CEE001 signalled' );
        END;
      ELSE  DO;
         DISPLAY( 'CEESGL  failed with msg '
            || FC.MsgNo );
         STOP;
         END;

   END PLISGL;
```

CEESICLR—Bit Clear (Manipulation Routine)

CEESICLR returns a copy of its *parm1* input, but with a single bit selectively set to 0. Bits are numbered from right to left, starting from 0.

Syntax



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 input is a 32-bit integer in the range 0 ≤ *parm2* ≤ 31.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

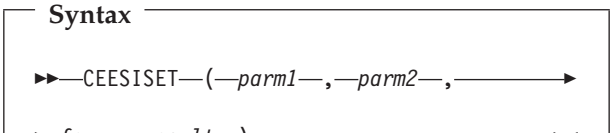
The following symbolic conditions can result from this service:

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 (Manipulation Routine)

CEESISET returns a copy of its *parm1* input, but with a single bit selectively set to 1. Bits are numbered from right to left, starting from 0.

Syntax



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 input is a 32-bit integer in the range 0 ≤ *parm2* ≤ 31.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

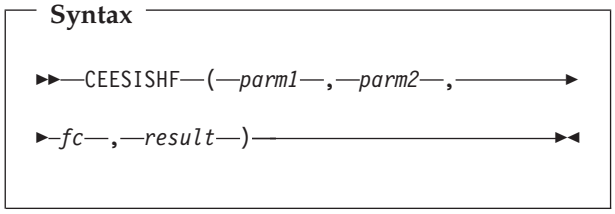
The following symbolic conditions can result from this service:

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 (Manipulation Routine)

CEESISHF returns a copy of its *parm1* input, right- or left-shifted by the number of bits indicated by *parm2*. Bits are numbered from right to left, starting from 0.

Syntax



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 input is a 32-bit integer in the range – 32 ≤ *parm2* ≤ 32.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

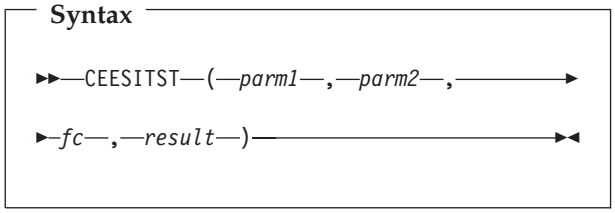
result (output)
The result of the Bit Shift routine. The output is a 32-bit integer whose value depends upon the value of *parm2*:
- If *parm2* ≥ 0, result is a copy of *parm1* shifted left by *parm2* bits.
- If *parm2* < 0, result is a copy of *parm1* shifted right by |*parm2*| bits.

In either case, vacated bits are set to 0.

CEESITST—Bit Test (Manipulation Routine)

CEESITST selectively tests a bit in its *parm1* input to determine if the bit is on. Bits are numbered from right to left, starting from 0.

Syntax



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 input is a 32-bit integer in the range 0 ≤ *parm2* ≤ 31.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

result (output)
The result of the Bit Test routine. The output is a 32-bit integer with value:
- 1, if bit number *parm2* in *parm1* is 1
- 0, if bit number *parm2* in *parm1* is 0

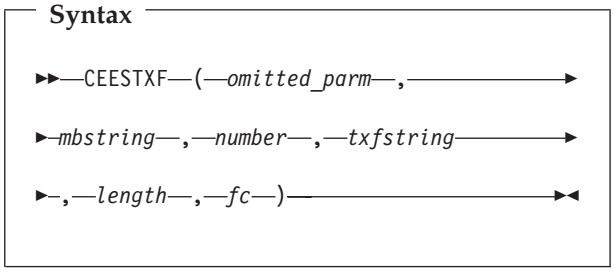
Bits are counted from the right.

CEESTXF—Transform String Characters into Collation Weights

CEESTXF, analogous to the C language function `strxfrm()`, 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. CEESTXF also returns the length of the transformed string.

CEESTXF is sensitive to the locales set by `setlocale()` or CEESETL, not to the LE/VSE settings from COUNTRY or CEE5CTY.

Syntax



omitted_parm
This parameter is reserved for future use. In C and PL/I, which allow you to omit parameters, it should be omitted. In COBOL, it should be coded as a dummy parameter. For more information about how to code an omitted parameter in C and PL/I, see "C Syntax" on page 58 and "PL/I Syntax" on page 59, respectively.

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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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.

For More Information

- For more information about the CEESETL callable service, see "CEESETL—Set Locale Operating Environment" on page 191.
- For more information on `setlocale()` and `strxfrm`, see *LE/VSE C Run-Time Library Reference* and
- For an explanation of the COUNTRY run-time option, see "COUNTRY" on page 26.
- For an explanation of the CEE5CTY callable service, see "CEE5CTY—Set Default Country" on page 65.

Examples

- **C Example**

  For C routines, the `strxfrm()` function should be used instead of the CEESTXF service. For details, refer to the *LE/VSE C Run-Time Library Reference*.
- **COBOL Example**

```
 CBL LIB,APOST,RMODE(ANY)
*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.                           *
************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  COBSTXF.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
* OMIT is a dummy parameter used across LE call.
 01  OMIT           COMP-2.
 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 OMIT, 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.
```

- **PL/I Example**

```
*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;

   /* 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 DL/I

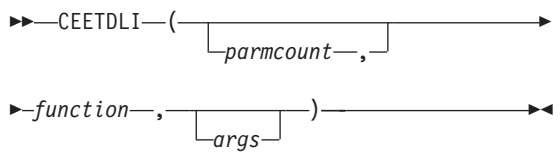
CEETDLI provides an interface to DL/I DOS/VS facilities. In assembler, COBOL, PL/I, and C, you can also invoke DL/I by using the following interfaces:

- In assembler, the ASMTDLI interface
- In COBOL, the CBLTDLI interface
- In PL/I, the PLITDLI interface
- In C, the CTDLI interface, a `ctdli()` function call

CEETDLI performs essentially the same functions as these interfaces, but is language independent.

The names CEETDLI, ASMTDLI, CBLTDLI, CTDLI, and PLITDLI are all interpreted as DL/I interfaces. If you are currently using them in any other way in your application, you must change them.

Syntax



parmcount
A fullword integer specifying the total number of arguments, excluding *parmcount*, for the CEETDLI call. This parameter is required for PL/I. For other languages, this option usually is not needed and can be omitted; it is supported for compatibility with earlier interface modules.

function
The DL/I function that you want to perform. The possible values for this field are defined by DL/I, not LE/VSE.

args
Arguments that you pass to DL/I. You cannot pass run-time options as CEETDLI arguments. You cannot alter the settings of run-time options when invoking DL/I facilities. The order and meaning of the arguments is defined by DL/I, not LE/VSE.

For More Information

- For more information about CEETDLI in the context of other DL/I interfaces, and in the context of DL/I condition handling, see *LE/VSE Programming Guide*.
- For information about the DL/I functions and argument parameters you can specify in CEETDLI, see *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*

Examples

- **C Example**

```
/*Module/File Name: EDCTDLI   */

#pragma runopts(env(DLI),plist(OS))
#include <ims.h>
#include <leawi.h>
#include <stdio.h>

/* ------------------------------------------------ */
/*                                                  */
/*   Function: Use CEETDLI - interface to DL/I      */
/*                           from C.                */
/*                                                  */
/*   In this example, a call is made to CEETDLI     */
/*   to interface to DL/I for an DL/I service.      */
/*                                                  */
/* ------------------------------------------------ */
/*                ENTRY POINT                       */
/* ------------------------------------------------ */

main() {

  typedef struct {PCB_STRUCT(150)} PCB_TYPE;
  PCB_TYPE *pcb_1;
  static char func_GN[4] = "GN  ";
  char seg_ioarea[160];
  pcb_1 = (PCB_TYPE *)(__pcblist[0]);

  CEETDLI(func_GN,pcb_1,seg_ioarea);
  printf("Status=%2.2s IOArea=%s\n",
         pcb_1->stat_code, seg_ioarea);
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *Module/File Name: IGZTTDLI
      ***********************************************/
      **                                           */
      ** Function: Use CEETDLI - interface to DL/I  */
      **                         from COBOL.       */
      **                                           */
      ** In this example, a call is made to CEETDLI */
      ** to interface to DL/I for an DL/I service.  */
      **                                           */
      ***********************************************/
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBL2DLI.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  FUNC-GN             PIC X(4) VALUE 'GN  '.
       01  SEG-IOAREA          PIC X(160).
       LINKAGE SECTION.
       01  CUST-L-PCB.
           02  CUST-DBD-NAME     PIC X(8).
           02  CUST-SEG-LEVEL    PIC XX.
           02  CUST-STAT-CODE    PIC XX.
           02  CUST-PROC-OPT     PIC XXXX.
           02  FILLER            PIC S9(5)    COMP.
           02  CUST-SEG-NAME     PIC X(8).
           02  CUST-LEN-KFB      PIC S9(5)    COMP.
           02  CUST-NU-SENSEG    PIC S9(5)    COMP.
           02  CUST-KEY-FBCK     PIC X(150).
      ** ENTRY POINT:
       PROCEDURE DIVISION.
           ENTRY 'DLITCBL' USING CUST-L-PCB.
           CALL 'CEETDLI' USING FUNC-GN, CUST-L-PCB,
                                SEG-IOAREA.
           DISPLAY 'Status of DL/I function = '
                   CUST-STAT-CODE.
           GOBACK.
```

- **PL/I Example**

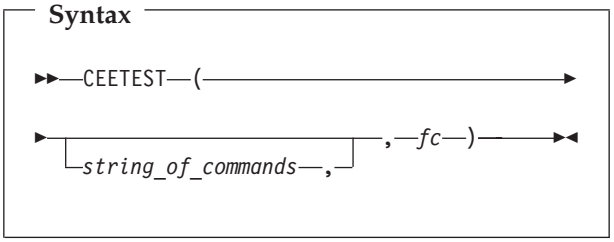
```
*PROCESS MACRO SYSTEM(DLI);
 /*Module/File Name: IBMTDLI                    */
 /* ------------------------------------------- */
 /*                                             */
 /* Function: Use CEETDLI - interface to DL/I   */
 /*                         from PL/I.          */
 /*                                             */
 /* In this example, a call is made to CEETDLI  */
 /* to interface to DL/I for an DL/I service.   */
 /*                                             */
 /* ------------------------------------------- */
 /*              ENTRY POINT                    */
 /* ------------------------------------------- */
 PLI2DLI: PROCEDURE(DB_PTR_CUST)
          OPTIONS(MAIN NOEXECOPS);

 %INCLUDE   CEEIBMAW;
 %INCLUDE   CEEIBMCT;

 DCL   FUNC_GN                CHAR(4) INIT('GN  ');
 DCL   THREE                  FIXED BIN (31,0)  INIT (3);

 DCL   DB_PTR_CUST            PTR;
 DCL   1 CUST_PCB             BASED (DB_PTR_CUST),
         2  CUST_DBD_NAME     CHAR (8),
         2  CUST_SEG_LEVEL    CHAR (2),
         2  CUST_STAT_CODE    CHAR (2),
         2  CUST_PROC_OPT     CHAR (4),
         2  FILLER            FIXED BIN (31,0),
         2  CUST_SEG_NAME     CHAR (8),
         2  CUST_LEN_KFB      FIXED BIN (31,0),
         2  CUST_NU_SENSEG    FIXED BIN (31,0),
         2  CUST_KEY_FBCK     CHAR (150);

 DCL 1 SEG_IOAREA             CHAR(160);

 CALL CEETDLI (THREE, FUNC_GN, CUST_PCB, SEG_IOAREA);
 PUT SKIP LIST('Status of DL/I function = '
               || CUST_STAT_CODE);

 RETURN;

 END PLI2DLI;
```

CEETEST—Invoke Debug Tool

CEETEST invokes a debug service such as Debug Tool for VSE/ESA.

Debug Tool for VSE/ESA supports debugging of LE/VSE, except for some restrictions noted in *Debug Tool for VSE/ESA User's Guide and Reference* If you want to invoke another interactive debug service, refer to the appropriate user's guide.

Syntax



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE2F2	3	2530	A debug tool was not available.

For More Information

- If you are using CEETEST to invoke Debug Tool for VSE/ESA and need more information about how to create a Debug Tool for VSE/ESA command list, refer to *Debug Tool for VSE/ESA User's Guide and Reference*

Examples

- **C Example**

```
/*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 */
  /* .
     .
     . */
}
```

- **COBOL Example**

```
CBL LIB,APOST
      *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.
```

- **PL/I Example**

```
*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;

    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 135.

Chapter 4. LE/VSE Math Services

LE/VSE math services provide standard math computations. You can call them from LE/VSE-conforming languages or from assembler routines by using the call interface (defined below and shown throughout this chapter), or the syntax specific to the HLL of your application.

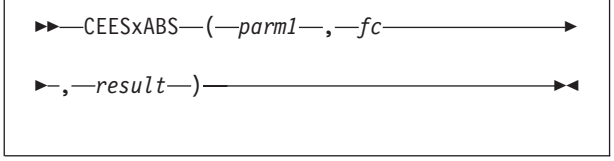
This chapter contains the following sections:

For a list of LE/VSE math services, see Chapter 1, "LE/VSE Quick Reference Tables," on page 1.

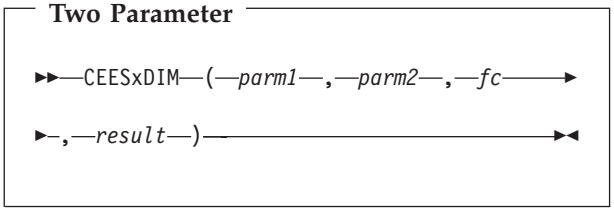
Call Interface to Math Services

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 Type and parm2 Type." 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



Two Parameter



Parameter Types: parm1 Type and parm2 Type

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 CEESx 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 [1]
E 64-bit double floating-point complex number [2]
R 128-bit extended floating-point complex number [3]

LE/VSE math services expect normalized input.

In the routines described in this chapter, the output range [4] for complex-valued functions can be determined from the input range.

For information about the data types supported by each LE/VSE-conforming HLL, see "Data Type Definitions" on page 60.

Feedback Code Parameter (fc)

fc 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

1. This parameter type consists of a real part and an imaginary part, each of which is a 32-bit single floating-point number.
2. This parameter type consists of a real part and an imaginary part, each of which is a 64-bit double floating-point number
3. This parameter type consists of a real part and an imaginary part, each of which is a 128-bit extended floating-point number.
4. 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.

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.

C, COBOL, and PL/I offer built-in math services that you can also use under LE/VSE. See *LE/VSE C Run-Time Programming Guide*, *IBM COBOL for VSE/ESA Language Reference* , or *IBM PL/I for VSE/ESA Language Reference* for a description of these functions.

Sample Calls to Math Services

This section contains sample calls to LE/VSE math services from C, COBOL, and PL/I.

C Call to CEESSLOG—Logarithm Base e

```
#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 Call to CEESSLOG—Logarithm Base e

```
⋮
    77   ARG1RS  COMP-1.
    77   FBCODE  PIC X(12).
    77   RESLTRS COMP-1.

         CALL "CEESSLOG" USING ARG1RS , FBCODE , RESLTRS.
⋮
```

PL/I Call to CEESSLOG—Logarithm Base e

```
⋮
DCL ARG1 RESULT REAL FLOAT DEC (6);
DCL FC   CHARACTER (12);
CALL CEESSLOG (ARG1, FC, RESULT)
⋮
```

In this chapter "**" means "raise to the power of". Therefore, 10**2 means that 10 is raised to the second power.

Math Services

This section contains an alphabetical list of the LE/VSE math services.

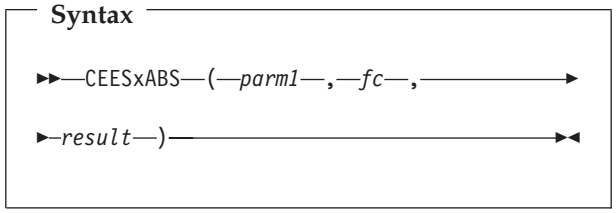
CEESxABS—Absolute Value

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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE1V9	1	2025	An underflow has occurred in math routine *routine-name*.

result (output)
The result of the absolute value routine. The output range is the non-negative numbers ≤ 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))

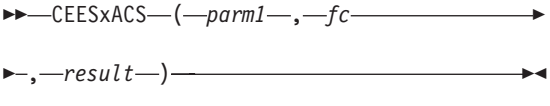
CEESxACS—Arccosine

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



parm1 (input)
The input to the arccosine routine. The input range is |*parm1*| ≤ 1.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the argument was greater than *limit* in math routine *routine-name* .

result (output)
The result, in radians, of the arccosine routine.

The output range is 0 ≤ *result* ≤ pi.

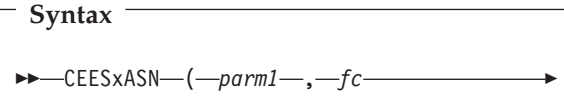
CEESxASN—Arcsine

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



parm1 (input)
The input to the arcsine routine. The input range is |*parm1*| ≤ 1.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the argument 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

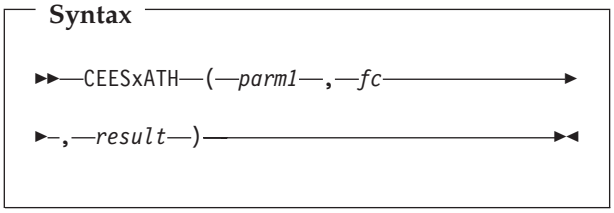
CEESxATH returns the hyperbolic arctangent of the parameter by using the equation *result* = (tanh**(-1))*parm1*.

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 (see note) 128-bit extended floating-point number
CEESTATH 32-bit single floating-point complex number
CEESEATH 64-bit double floating-point complex number
CEESRATH (see note) 128-bit extended floating-point complex number

Note: Math services CEESQATH and CEESRATH are not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



parm1 (input)
The input to the hyperbolic arctangent routine.

The input range for real variables is |*parm1*| < 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
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* .
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

result (output)
The result of the hyperbolic arctangent routine. The output range for functions of real variables is |*result*| ≤ 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

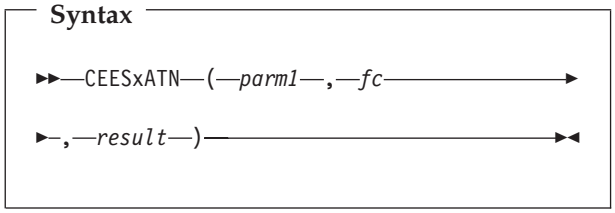
CEESxATN returns the arctangent of the parameter by using the equation *result* = arctan(*parm1*).

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 (see note) 128-bit extended floating-point complex number

Note: Math service CEESRATN is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



parm1 (input)
The input to the arctangent routine. The input range for real variables is not restricted. The input range for complex variables is *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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1V6	2	2022	The value of the argument was plus or minus *limit* in math routine *routine-name* .
CEE1V9	1	2025	An underflow has occurred in math routine *routine-name*.
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

result (output)
The result, in radians, of the arctangent routine. The output range for functions of real variables is $|result| < pi/2$.

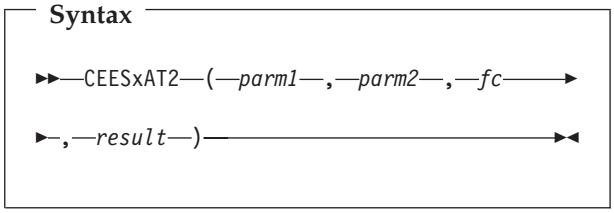
CEESxAT2—Arctangent2

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 pi, inclusive. 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



parm1 (input)
The first input to the arctangent2 routine. The input range is *parm1* cannot equal 0 if *parm2* equals 0.

parm2 (input)
The second parameter to the arctangent2 routine. The input range is *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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UU	2	2014	Both arguments were equal to *limit* in math routine *routine-name* .
CEE1V9	1	2025	An underflow has occurred in math routine *routine-name*.

result (output)
The result, in radians, of the arctangent2 routine. The output range is $|result| \leq pi$.

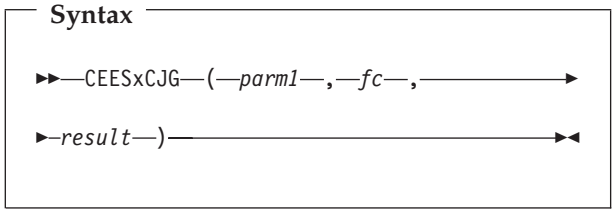
CEESxCJG—Conjugate of Complex

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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

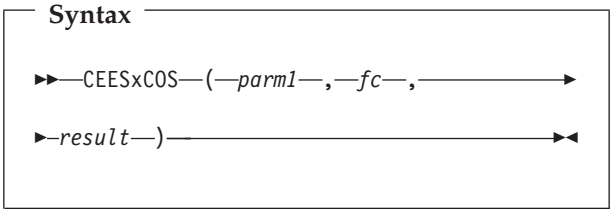
result (output)
The result of the conjugate of complex routine.

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



parm1 (input)
The type is determined by the fifth character of the service name. The input range for real variables is
| *parm1* | < (2**18 • pi), for single floating-point numbers
| *parm1* | < (2**50 • pi), for double floating-point numbers
| *parm1* | < 2**100, for extended floating-point numbers

For complex functions, the input range differs for the imaginary and real parts of the input. For the imaginary part, the input range is | Im(*parm1*) | < 174.673. For the real part of the input, the input range is
|Re(*parm1*)| < (2**18 • pi), for single floating-point complex numbers
|Re(*parm1*)| < (2**50 • pi), for double floating-point complex numbers
|Re(*parm1*)| < 2**100, for extended floating-point complex numbers

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UT	2	2013	The absolute value of the imaginary part of the argument was greater than *limit* in math routine *routine-name* .
CEE1V1	2	2017	The absolute value of the argument was greater than or equal to *limit* in math routine *routine-name* .
CEE1V3	2	2019	The absolute value of the real part of the argument 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* | ≤ 1.

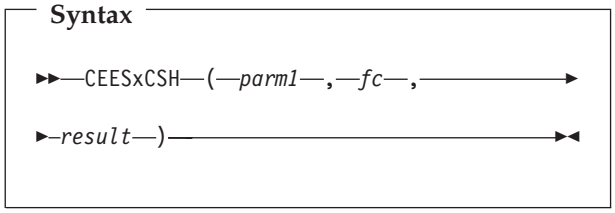
CEESxCSH—Hyperbolic Cosine

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



parm1 (input)
The input to the hyperbolic cosine routine.

The input range for the functions of real variables is |*parm1*| < 175.366.

For complex functions, the input range differs for the imaginary and real parts of the input. For the real part of complex numbers, the input range is |Re(*parm1*)| < 175.366. For the imaginary part of the input, the input range is
|Im(*parm1*)| < (2**18 • pi), for single floating-point complex numbers
|Im(*parm1*)| < (2**50 • pi), for double floating-point complex numbers
|Im(*parm1*)| < 2**100, for extended floating-point complex numbers

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the argument 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 is 1 ≤ *result* ≤ 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))

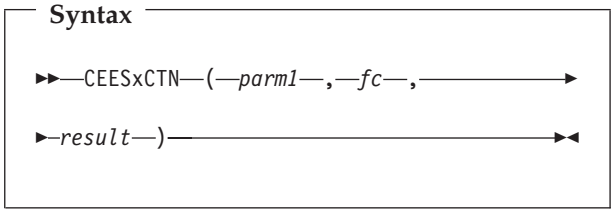
CEESxCTN—Cotangent

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



parm1 (input)
The input, in radians, into the cotangent routine. The input range varies, depending on the precision of *parm1*:
Single-precision
|*parm1*| < 2**18 • pi
Double-precision
|*parm1*| < 2**50 • pi

Extended-precision
|*parm1*| < 2**100 • pi

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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UI	2	2002	The argument 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 argument was greater than or equal to *limit* in math routine *routine-name* .

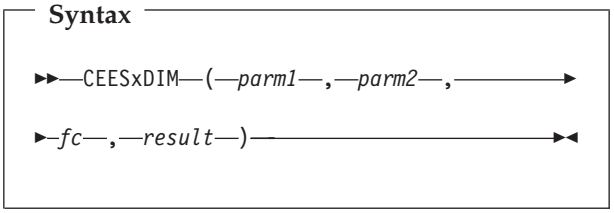
result (output)
The result of the cotangent routine. The output range is *result* ≤ Omega. Omega varies depending on the precision of *parm1*:
For single-precision routines, Omega = (16**63)(1-16**(-6))
For double-precision routines, Omega = (16**63)(1-16**(-14))
For extended-precision routines, Omega = (16**63)(1-16**(-28))

CEESxDIM—Positive Difference

CEESxDIM returns the positive difference between two numbers by using one of the following equations:
if *parm1* > *parm2*, then *result* = *parm1* - *parm2*
if *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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 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))

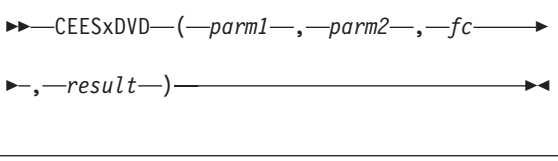
CEESxDVD—Floating-Point Complex Divide

CEESxDVD performs the mathematical function of floating-point complex divide by using the equation: *result* = *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 (see note)
128-bit extended floating-point complex number

Note: Math service CEESRDVD is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



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 passed by reference indicating the result of the service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

result (output)
The result of the floating-point complex divide routine.

CEESxERC—Error Function Complement

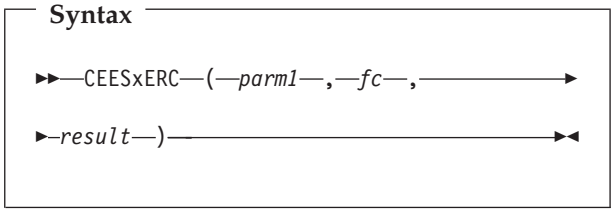
CEESxERC calculates the error function complement. If necessary, see your printed book for equation.

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 (see note)
128-bit extended floating-point number

Note: Math service CEESQERC is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

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. If necessary, see your printed book for equation.

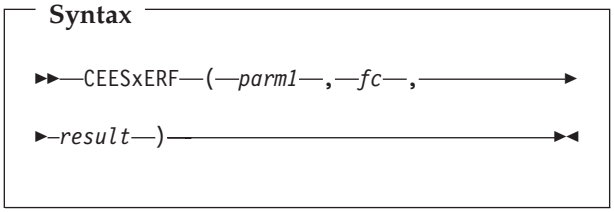
The following routines are provided for the various data types supported:

CEESSERF 32-bit single floating-point number

CEESDERF 64-bit double floating-point number
CEESQERF (see note) 128-bit extended floating-point number

Note: Math service CEESQERF is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

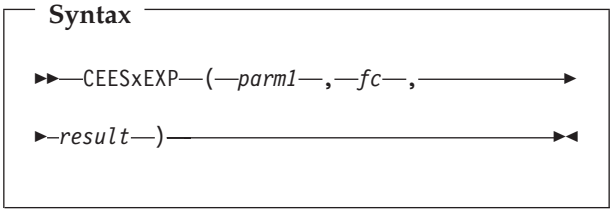
result (output)
The result of the error function. The output range is | *result* | < 1.

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



parm1 (input)
The input to the exponential base e routine.

For complex functions, the input range differs for the imaginary and real parts of the input. For the real part of complex numbers, the input range is |Re(*parm1*)| < 174.673. For the imaginary part of the input, the input range is
|Im(*parm1*)| < (2**18 • pi), for single floating-point complex numbers
|Im(*parm1*)| < (2**50 • pi), for double floating-point complex numbers
|Im(*parm1*)| < 2**100, for extended floating-point complex numbers

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UP	2	2009	The value of the real part of the argument was greater than *limit* in math routine *routine-name* .
CEE1UR	2	2011	The argument was greater than *limit* in math routine *routine-name* .
CEE1UT	2	2013	The absolute value of the imaginary part of the argument was greater than *limit* in math routine *routine-name* .

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE1UV	2	2015	The absolute value of the imaginary part of the argument was greater than or equal to *limit* in math routine *routine-name*.
CEE1V9	1	2025	An underflow has 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 < *result* ≤ 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))

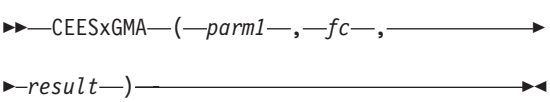
CEESxGMA—Gamma Function

CEESxGMA performs the mathematical gamma function. If necessary, see your printed book for equation.

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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UL	2	2005	The value of the argument was outside the valid range *range* in math routine *routine-name* .

result (output)
The result of the gamma function. The output range is 0.88560 ≤ *result* 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))

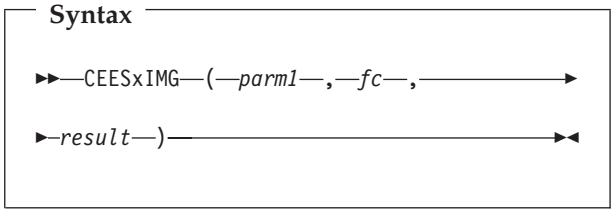
CEESxIMG—Imaginary Part of Complex

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



parm1 (input)
The input to the math service. Any complex number can be used as input.

fc (output)
A 12-byte feedback code passed by reference indicating the result of the service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

result (output)
The result of the math service. |*result*| ≤ 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))

CEESxINT—Truncation

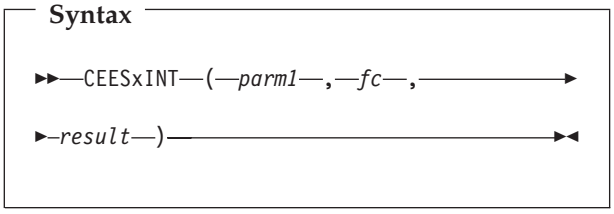
CEESxINT returns the truncated value of the parameter by using the equation: *result* = (sign of *parm1*) • *n*, where *n* = | *parm1* |.

| *parm1* | = | *m* |, where *m* is the greatest integer satisfying the relationship | *m* | ≤ | *parm1* |, and the result is expressed as a floating-point number.

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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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*| ≤ 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))

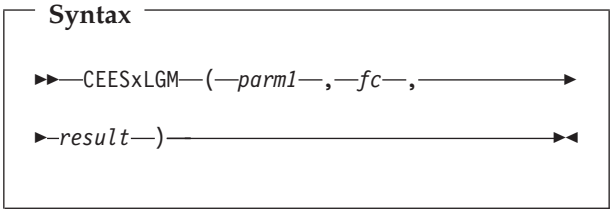
CEESxLGM—Log Gamma

CEESxLGM performs the mathematical function of log Gamma. If necessary, see your printed book for equation.

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



parm1 (input)
The input to the log gamma routine. The input range is *parm1* < (4.2913 • 10**73).

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE1UL	2	2005	The value of the argument was outside the valid range *range* in math routine *routine-name* .

result (output)
The result of the log gamma routine. The output range is -0.12149≤*result*≤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))

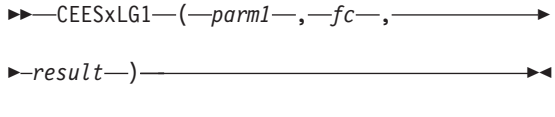
CEESxLG1—Logarithm Base 10

CEESxLG1 returns the logarithm base 10 of *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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1US	2	2012	The argument 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 -78.268≤*result*≤75.859.

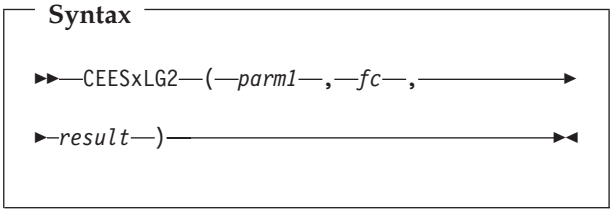
CEESxLG2—Logarithm Base 2

CEESxLG2 performs the mathematical function logarithm base 2 on *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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1US	2	2012	The argument 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 –260≤*result*≤252.

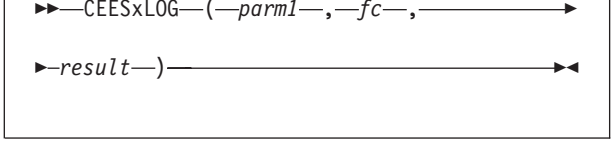
CEESxLOG—Logarithm Base e

CEESxLOG performs the mathematical function logarithm base e on *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



parm1 (input)
The input to the log base e routine. The input range for reals is *parm1* > 0. The input range for complex numbers is *parm1* not equal to 0.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1US	2	2012	The argument was less than or equal to *limit* in math routine *routine-name* .
CEE1V2	2	2018	The real and imaginary parts of the argument 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 -180.218 ≤ *result* ≤ 174.673.

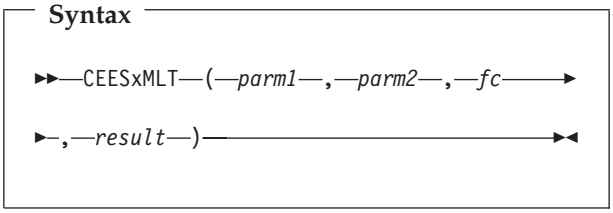
CEESxMLT—Floating-Point Complex Multiply

CEESxMLT performs the mathematical function floating-point complex multiply by using the equation *result* = *parm1* • *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



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 passed by reference indicating the result of the service.

The following symbolic conditions can result from this service:

Symbolic Feedback 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* = *parm1*(modulo *parm2*).

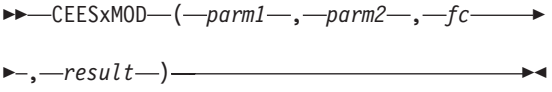
The expression *parm1* (modulo *parm2*) is defined as *parm1* – [(*parm1*/*parm2*) • *parm2*], with the brackets indicating an integer part, that is, the largest integer whose magnitude does not exceed the magnitude of *parm1*/*parm2* is used. The sign of the integer is the same as the sign of *parm1* • *parm2*.

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 (see note) 128-bit extended floating-point number

Note: Math service CEESQMOD is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



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* cannot equal 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

result (output)
The result of the mod routine. The output range is |*result*| ≤ Omega. Omega varies depending on the precision of *parm1* and *parm2*:
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*.

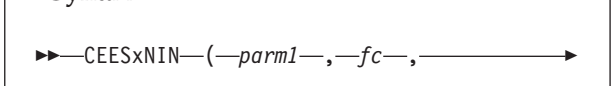
If *parm1* ≥ 0, then *n* = [| *parm1* + .5 |]. If *parm1* < 0, then *n* = [| *parm1* - .5 |].

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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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*.

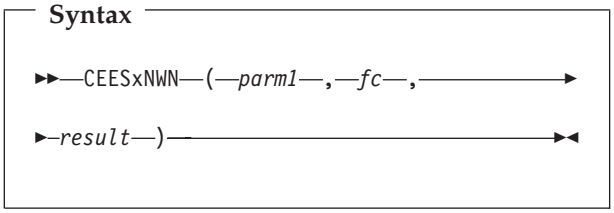
If *parm1* ≥ 0, then *v* = | *parm1* + .5 |. If *parm1* < 0, then *v* = [| *parm1* - .5 |].

v = | *m* |, where *m* is the greatest integer satisfying the relationship | *m* | ≤ | *parm1* +.5 |, or | *m* | ≤ | *parm1* -.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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 is | *result* | ≤ 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))

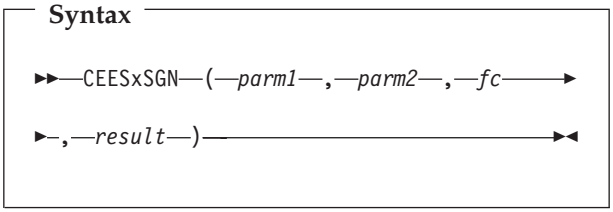
CEESxSGN—Transfer of Sign

CEESxSGN performs the mathematical function transfer of sign by using either of the two equations: *result* = |*parm1*| if *parm2* ≥ 0 or *result* = -|*parm1*| 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



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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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*| ≤ 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))

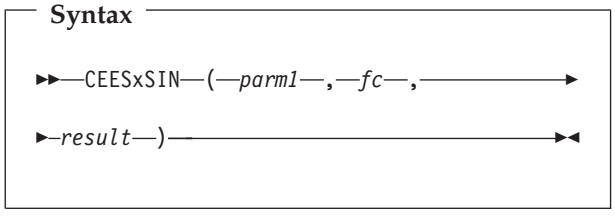
CEESxSIN—Sine

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



parm1 (input)
The input, in radians, to the sine routine. For real functions, the input range differs depending on the precision of *parm1*:
|*parm1*| < (2**18 • pi), for single-precision numbers
|*parm1*| < (2**50 • pi), for double-precision numbers
|*parm1*| < 2**100, for extended-precision numbers

For complex functions, the input range differs for the imaginary and real parts of the input. For the imaginary part of complex numbers, the input range is |Im(*parm1*)| < 174.673.

For the real part, it is
|Im(*parm1*)| < (2**18 • pi), for single floating-point complex numbers
|Im(*parm1*)| < (2**50 • pi), for double floating-point complex numbers
|Im(*parm1*)| < 2**100, for extended floating-point complex numbers

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UT	2	2013	The absolute value of the imaginary part of the argument was greater than *limit* in math routine *routine-name* .
CEE1V1	2	2017	The absolute value of the argument was greater than or equal to *limit* in math routine *routine-name* .
CEE1V3	2	2019	The absolute value of the real part of the argument 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 -1 ≤ *result* ≤ 1.

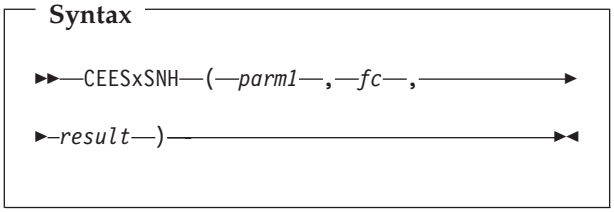
CEESxSNH—Hyperbolic Sine

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



parm1 (input)
The input to the hyperbolic sine routine. Input range for reals is |*parm1*| < 175.366.

For complex functions, the input range differs for the imaginary and real parts of the input. For the real part, the input range is |Re(*parm1*)| < 174.673. The input range of the imaginary part differs depending on the precision of *parm1*:
|Im(*parm1*)| < (2**18 • pi), for single floating-point complex numbers
|Im(*parm1*)| < (2**50 • pi), for double floating-point complex numbers
|Im(*parm1*)| < 2**100, for extended floating-point complex numbers

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1V0	2	2016	The absolute value of the argument 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* | ≤ 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))

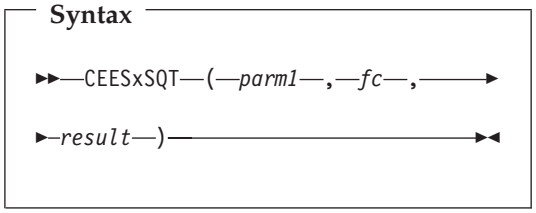
CEESxSQT—Square Root

CEESxSQT returns the square root of *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 (see note) 128-bit extended floating-point complex number

Note: Math service CEESRSQT is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



parm1 (input)
The input to the square root routine. The input range for real number functions is *parm1* ≥ 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UQ	2	2010	The argument was less than *limit* in math routine *routine-name* .
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

result (output)
The result of the square root routine. The output range for functions of real variables is 0 ≤ *result* ≤ square root of 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))

CEESxTAN—Tangent

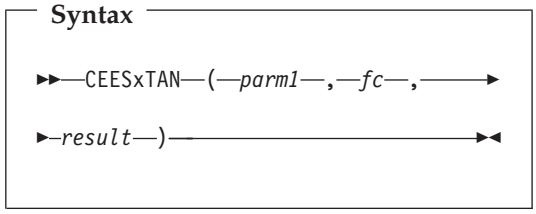
CEESxTAN returns the tangent of the parameter by using the equation *result* = tan (*parm1*).

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 (see note) 128-bit extended floating-point complex number

Note: Math service CEESRTAN is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



parm1 (input)
The input, in radians, to the tangent routine. For real functions, the input range differs depending on the precision of *parm1*:
|*parm1*| < (2**18 • pi), for single-precision numbers
|*parm1*| < (2**50 • pi), for double-precision numbers
|*parm1*| < 2**100, for extended-precision numbers

Also, for extended as well as complex functions, *parm1* 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.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE1UI	2	2002	The argument 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 argument was greater than or equal to *limit* in math routine *routine-name* .
CEE1V9	1	2025	An underflow has occurred in math routine *routine-name*.
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

result (output)
The result of the tangent routine. The output range for functions of real variables is

| *result* | ≤ Ω. The output range is the non-negative numbers ≤ 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))

CEESxTNH—Hyperbolic Tangent

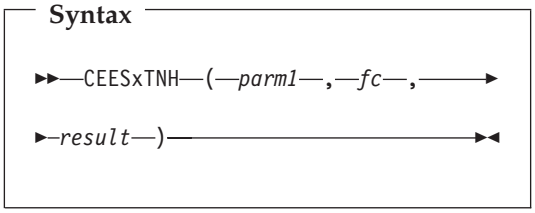
CEESxTNH performs the mathematical function hyperbolic tangent by using the equation: *result* = tanh (*parm1*).

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 (see note) 128-bit extended floating-point complex number

Note: Math service CEESRTNH is not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



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 • *i*, where *i* = sqrt of -1.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.

The following symbolic conditions can result from this service:

Symbolic Feedback Code	Severity	Message Number	Message Text
CEE000	0	—	The service completed successfully.
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

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 (see note) 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 (see note) 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

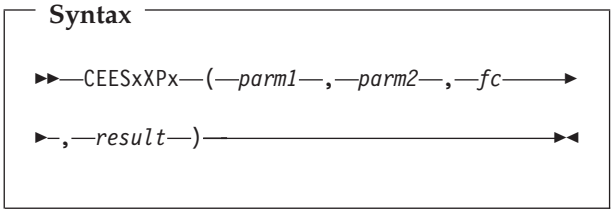
CEESTXPT 32-bit single floating-point complex raised to a 32-bit single floating-point complex

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

Note: Math services CEESQXPI and CEESRXPI are not supported on systems running in 370 mode (for example, VSE/ESA running on a 370-mode machine, or VSE/ESA running as a 370-mode guest system under VM).

Syntax



parm1 (input)

The input for the base of the exponentiation routine. The input range for functions of real variables is

- If *parm1* = 0, then *parm2* > 0.
- If *parm1* is a 32-bit number and *parm1* < 0, then |*parm1*| ≤ ((16**6) - 1) and *parm2* = a whole number.
- If *parm1* is a 64-bit number and *parm1* < 0, then |*parm1*| ≤ ((16**14) - 1) and *parm2* = a whole number.
- If *parm1* is a 128-bit number and *parm1* < 0, then |*parm1*| ≤ ((16**28) - 1) and *parm2* = a whole number

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.

The following symbolic conditions can result from this service:

Symbolic Feedback 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 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 argument 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* .
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 argument combination of Y*log2(X) generated a number greater than or equal to *limit* in math routine *routine-name* .
CEE34K	3	3220	A math service attempted to execute an extended-precision floating-point-divide machine instruction while running in 370 mode.

result (output)

The result of the exponentiation routine. The output range for functions of real variables is

| *result* | ≤ 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))

Examples of Math Services

This section contains representative calls to math services from each supported language.

C Math Service Example

The following are C examples.

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

The following are COBOL examples.

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.
```

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

The following are PL/I examples.

Modular Arithmetic and Log Base e (CEESIMOD and CEESSLOG)

```
*PROCESS MACRO;
 PLIMATH: PROC OPTIONS(MAIN);

 %INCLUDE CEEIBMAW;
 %INCLUDE CEEIBMCT;

    DCL (ARG1,RESULT) REAL FLOAT BINARY(109);
    DCL (ARGM1,ARGM2,RES2) REAL FLOAT BINARY(109);
    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 log base e routine, which has    */
   /* only one input parameter              */
    UNSPEC(ARG1) = '4C61A4585F8951913EA836EB4E981388'BX;
    CALL CEESQLOG (ARG1, FC, RESULT);

    IF  ¬ FBCHECK( FC, CEE000) THEN
        PUT SKIP LIST
           ( 'Error occurred in call to CEESQLOG.');

   /* Call modular arithmetic routine,      */
   /* which has two input parameters        */
    UNSPEC(ARGM1) = '41F0000000000000330000000000000'BX;
    UNSPEC(ARGM2) = 'C180000000000000B300000000000000'BX;
    CALL CEESQMOD (ARGM1, ARGM2, FC, RES2);

    IF  ¬ FBCHECK( FC, CEE000) THEN
        PUT SKIP LIST
           ( 'Error occurred in call to CEESQMOD.');

 END PLIMATH;
```

Double-Precision Complex Tangent (CEESETAN)

```
*PROCESS MACRO;
 TRYETAN:  PROCEDURE OPTIONS( MAIN );

 %INCLUDE CEEIBMAW;
 %INCLUDE CEEIBMCT;
```

EXAMPLES

```
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);
```

```
   DECLARE PARM1        COMPLEX FLOAT BINARY(53);
   DECLARE RESULT       COMPLEX FLOAT BINARY(53);

   PARM1 = COMPLEX(7,1.1);
   CALL CEESETAN ( PARM1, FC, RESULT);
   IF  ¬ FBCHECK( FC, CEE000) THEN
         PUT SKIP LIST( 'Error in call to CEESETAN.');
   ELSE
         PUT SKIP LIST( 'Result is ' || RESULT);
END TRYETAN;
```

Appendix A. IBM-Supplied Country/Region Code Defaults

Table 32 contains the currency symbols and default picture strings for the *country_code* parameters of the COUNTRY run-time option (which cover both countries and regions within those countries) and the national language support callable services.

Note: In the table, some currency symbols are shown in hexadecimal representation. How this is displayed depends on the codeset that is used. Where an applicable codeset for the country/region is used, this represents the national currency symbol.

Table 32. Default Currency and Picture Strings Based on Country / Region Setting

Country / Region	Code	Default Decimal Separator	Default Thousand Separator	Default Currency Symbol	Default Time Picture String	Default Date Picture String	Default Date and Time Picture String
Afghanistan	AF	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Albania	AL	,	.	Lek	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Algeria	DZ	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Andorra	AD	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Angola	AO	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Antigua and Barbuda	AG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Argentina	AR	,	.	A.	HH.MI.SS	DD/MM/YY	DD/MM/YY HH.MI.SS
Australia	AU	.	,	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Austria	AT	,	.	X'9F404040'	HH:MI:SS,999	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS,999
Bahamas	BS	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bahrain	BH	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Bangladesh	BD	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Barbados	BB	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Belgium	BE	,	.	X'9F404040'	HH:MI:SS,999	DD/MM/YY	DD/MM/YY HH:MI:SS,999
Benin	BJ	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bermuda	BM	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bolivia	BO	,	.	BS	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Bosnia/ Herzegovina	BA	,	.	Din (Dinar)	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Botswana	BW	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Brazil	BR	,	.	NCz$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Brunei Darussalam	BN	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Bulgaria	BG	,	.	Lv (Lev)	HH:MI:SS	YYYY-RRRZ-DD	YYYY-RRRZ-DD HH:MI:SS
Burkina Faso (Upper Volta)	BF	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Burma	BU	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Burundi	BI	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Cameroon	CM	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Canada	CA	.	,	$	HH:MI:SS.99	YY-MM-DD	YY-MM-DD HH:MI:SS.99
Cayman Islands	KY	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 32. Default Currency and Picture Strings Based on Country / Region Setting (continued)

Country / Region	Code	Default Decimal Separator	Default Thousand Separator	Default Currency Symbol	Default Time Picture String	Default Date Picture String	Default Date and Time Picture String
Central Africa Republic	CF	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Chad	TD	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Chile	CL	,	.	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Colombia	CO	,	.	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Congo	CG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Costa Rica	CR	,	.	c/.	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Croatia	HR	,	.	Din (Dinar)	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Cuba	CU	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Cyprus	CY	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Czech Republic	CZ	,	.	X'D247A240'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Czecho-slovakia	CS	,	.	X'D247A240'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Denmark	DK	,	.	kr	HH.MI.SS,99	DD-MM-YY	DD-MM-YY HH.MI.SS,99
Dominican Republic	DO	.	,	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Ecuador	EC	,	.	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Egypt	EG	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
El Salvador	SV	.	,	c/.	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Estonia	EE	,		Kr (Kroon)	HH:MI:SS	DD-MM-YYYY	DD-MM-YYYY HH:MI:SS
Ethiopia	ET	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Finland	FI	,		X'5A404040'	HH.MI.SS,999	DD.MM.YYYY	DD.MM.YYYY HH.MI.SS,99
France	FR	,		X'9F404040'	HH:MI:SS,9	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS,9
Gabon	GA	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Gambia	GM	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Germany	DE	,	.	X'9F404040'	HH:MI:SS	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS
Ghana	GH	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Greece	GR	,	.	X'FC404040'	HH:MI:SS.999	DD/MM/YY	DD/MM/YY HH:MI:SS.999
Guatemala	GT	.	,	Q	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Guinea	GN	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Guinea-Bissau	GW	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Guyana	GY	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Haiti	HT	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Honduras	HN	.	,	L.	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Hong Kong	HK	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Hungary	HU	,		FT	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Iceland	IS	,	.	kr	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
India	IN	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Indonesia	ID	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Iran	IR	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Iraq	IQ	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Ireland	IE	,	,	X'9F404040'	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS

Table 32. Default Currency and Picture Strings Based on Country / Region Setting (continued)

Country / Region	Code	Default Decimal Separator	Default Thousand Separator	Default Currency Symbol	Default Time Picture String	Default Date Picture String	Default Date and Time Picture String
Israel	IL	.	,	NIS	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Italy	IT	,	.	X'9F404040'	HH.MI.SS,999	DD/MM/YY	DD/MM/YY HH.MI.SS,999
Jamaica	JM	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Japan	JP	.	,	X'5B404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Jordan	JO	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Kenya	KE	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Korea, Republic of	KR	.	,	X'E0404040'	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Kuwait	KW	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Latvia	LV	,		Ls (Lats)	HH:MI:SS	YYYY-DD-RRRZ	YYYY-DD-RRRZ HH:MI:SS
Lebanon	LB	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Lesotho	LS	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Liberia	LR	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Libya	LY	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Liecht-enstein	LI	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Lithuania	LT	,	.	Lt (litas)	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Luxembourg	LU	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Macau	MO	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Macedonia, Former Yugoslav Republic of	MK	,	.	Den (Denar)	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Madagascar	MG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malawi	MW	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malaysia	MY	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mali	ML	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Malta	MT	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mauritania	MR	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mauritius	MU	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Mexico	MX	.	,	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Monaco	MC	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Morocco	MA	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Mozam-bique	MZ	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Namibia	NA	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Netherlands	NL	,	.	X'9F404040'	HH:MI:SS	DD-MM-YY	DD-MM-YY HH:MI:SS
Netherlands Antilles	AN	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
New Caledonia	NC	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
New Zealand	NZ	.	,	$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Nicaragua	NI	.	,	X'9F404040'	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Niger	NE	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Nigeria	NG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 32. Default Currency and Picture Strings Based on Country / Region Setting (continued)

Country / Region	Code	Default Decimal Separator	Default Thousand Separator	Default Currency Symbol	Default Time Picture String	Default Date Picture String	Default Date and Time Picture String
Norway	NO	,	.	kr	HH:MI:SS,999	DD.MM.YY	DD.MM.YY HH:MI:SS,999
Oman	OM	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Pakistan	PK	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Panama	PA	.	,	B/	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Papua New Guinea	PG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Paraguay	PY	,	.	Gs.	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
People's Republic of China	CN	.	,	X'5B404040'	HH:MI:SS	YYYY.MM.DD	YYYY.MM.DD HH:MI:SS
Peru	PE	.	,	I/.	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Philippines	PH	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Poland	PL	,	.	X'E99A4040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Portugal	PT	,	.	X'9F404040'	HH:MI:SS	DD-MM-YYYY	DD-MM-YYYY HH:MI:SS
Puerto Rico	PR	.	,	$	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Qatar	QA	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Romania	RO	,	.	Lei	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Russia	RU	,	.	Rub (Ruble)	HH:MI:SS	DD mmm. YYYY g.	DD mmm. YYYY g. HH:MI:SS
Saint Lucia	LC	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Saudi Arabia	SA	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Senegal	SN	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Seychelles	SC	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sierra Leone	SL	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Singapore	SG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Slovakia	SK	,	.	X'D247A240'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Slovenia	SI	,	.	tol (Tolar)	HH:MI:SS	DD.MM.YYYY	DD.MM.YYYY HH:MI:SS
Somalia	SO	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
South Africa	ZA	.		R	HHhMI:SS.999	YYYY-MM-DD	YYYY-MM-DD HHhMI:SS.999
Spain	ES	,	.	X'9F404040')	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Sri Lanka	LK	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sudan	SD	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Surinam	SR	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Swaziland	SZ	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Sweden	SE	,	.	kr	kl HH.MI.SS	YYYY-MM-DD	YYYY-MM-DD kl HH.MI.SS
Switzer-land	CH	,	.	Fr	HH,MI,SS	DD. Mmmmmmmmz YYYY	DD. Mmmmmmmmz YYYY HH,MI,SS
Syria	SY	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Taiwan	TW	.	,	$	HH:MI:SS.999	YY/MM/DD	YY/MM/DD HH:MI:SS.999
Tanzania	TZ	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Thailand	TH	.	,	X'70404040'	HH:MI:SS	DD/MM/YYYY	DD/MM/YYYY HH:MI:SS
Togo	TG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Trinidad and Tobago	TT	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Table 32. Default Currency and Picture Strings Based on Country / Region Setting (continued)

Country / Region	Code	Default Decimal Separator	Default Thousand Separator	Default Currency Symbol	Default Time Picture String	Default Date Picture String	Default Date and Time Picture String
Tunisia	TN	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Turkey	TR	,	.	TL	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Uganda	UG	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Union of Soviet Socialist Republics	SU	,	.	Rub	HH:MI:SS	DD mmm. YYYY g.	DD mmm. YYYY g. HH:MI:SS
United Arab Emirates	AE	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
United Kingdom	GB	.	,	X'5B404040'	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
United States	US	.	,	$	ZH:MI:SS AP	MM/DD/YY	MM/DD/YY ZH:MI:SS AP
Uruguay	UY	,	.	N$	HH:MI:SS	DD/MM/YY	DD/MM/YY HH:MI:SS
Vanuatu	VU	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Venezuela	VE	,	.	Bs.	ZH:MI:SS AP	DD/MM/YY	DD/MM/YY ZH:MI:SS AP
Western Samoa	WS	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Yemen	YE	,	.		HH:MI:SS	YYYY/MM/DD	YYYY/MM/DD HH:MI:SS
Yugoslavia	YU	,	.	Din	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zaire	ZR	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zambia	ZM	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Zimbabwe	ZW	,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS
Default		,	.	X'9F404040'	HH:MI:SS	YYYY-MM-DD	YYYY-MM-DD HH:MI:SS

Note:

In other versions of Language Environment:

- Country code CS was previously used for Czechoslovakia. Instead of CS you should use either the Czech Republic country code CZ, or the Slovakia country code SK.
- Country code DE was previously used for the Federal Republic of Germany. Now this 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.
- Country code SU was previously used for the Union of Soviet Socialist Republics. Instead of SU you should 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 LE/VSE 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	Notes
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/ZYY used with <JJJJ>, <CCCC> and <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 234.
<CCCC> <CCCCCCCC>	Chinese era name in DBCS characters	*MinKow* (X'0E4D8256CE0F') *ChuHwaMinKow* (X'0E4C845ADD4D8256CE0F')	Affects YY field: if <CCCC> specified, YY means the year within Chinese era, for example, 1988 equals Minkow 77. See example in Table 34 on page 234.
MM	2-digit month	01-12	
ZM	1- or 2-digit month	1-12	For output, leading zero suppressed. For input, ZM treated as MM.
RRRR RRRZ	Roman numeral month	Ibb-XIIb (Left justified)	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 (b) (unless Z specified) or truncated to match the number of Ms, up to 20.
Mmm	3-char month, mixed case	Jan-Dec	
MMMM...M	3-20 char mo., uppercase	JANUARYbb-DECEMBERb	
Mmmm...m	3-20 char mo., mixed case	Januarybb-Decemberb	
MMMMMMMMMZ	trailing blanks suppressed	JANUARY-DECEMBER	
Mmmmmmmmmz	trailing blanks suppressed	January-December	
DD	2-digit day of month	01-31	For output, leading zero is always suppressed. For input, ZD treated as DD.
ZD	1- or 2-digit day of mo.	1-31	
DDD	day of year (Julian day)	001-366	
HH	2-digit hour	00-23	
ZH	1- or 2-digit hour	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	tenths of a second	0-9	No rounding.
99	hundredths of a second	00-99	
999	thousandths of a second	000-999	
AP	AM/PM indicator	AM or PM	AP affects HH/ZH field. For input, source string always folded to uppercase. For output, AP generates uppercase and ap generates lowercase.
ap		am or pm	
A.P.		A.M. or P.M.	
a.p.		a.m. or p.m.	

Table 33. Picture Character Terms Used in Picture Strings for Date and Time Services (continued)

Picture Terms	Explanations	Valid Values	Notes
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	SUNDAYbbb-SATURDAYb	
Www...w	3-20 char day, mixed case	Sundaybbb-Saturdayb	
WWWWWWWWWZ	trailing blanks suppressed	SUNDAY-SATURDAY	
Wwwwwwwwwz	trailing blanks suppressed	Sunday-Saturday	
All others	delimiters	X'01'-X'FF' (X'00' reserved by LE/VSE)	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.

Table 34. Examples of Picture Strings Recognized by Date and Time Services

Picture Strings	Examples	Notes
YYMMDD	880516	
YYYYMMDD	19880516	
YYYY-MM-DD	1988-05-16	1988-5-16 would also be valid input.
<JJJJ> YY.MM.DD	*Showa* 63.05.16	*Showa* is a Japanese Era name. *Showa* 63 equals 1988.
<CCCC> YY.MM.DD	*MinKow* 77.05.16	*MinKow* is a Chinese Era name. *MinKow* 77 equals 1988.
MMDDYY	050688	
MM/DD/YY	05/06/88	
ZM/ZD/YY	5/6/88	
MM/DD/YYYY	05/06/1988	
MM/DD/Y	05/06/8	1-digit year format (Y) valid for output only
DD.MM.YY	09.06.88	
DD-RRRR-YY	09-VI -88	
DD MMM YY	09 JUN 88	
DD Mmmmmmmmm YY	09 June 88	
ZD Mmmmmmmmmz YY	9 June 88	Z suppresses zeros/blanks
Mmmmmmmmmz ZD, YYYY	June 9, 1988	
ZDMMMMMMMMzYY	9JUNE88	
YY.DDD	88.137	Julian date
YYDDD	88137	
YYYY/DDD	1988/137	
YYMMDDHHMISS	880516204229	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.
YYYYMMDDHHMISS	19880516204229	
YYYY-MM-DD HH:MI:SS.999	1988-05-16 20:42:29.046	
WWW, ZM/ZD/YY HH:MI AP	MON, 5/16/88 08:42 PM	
Wwwwwwwwwz, DD Mmm YYYY, ZH:MI AP	Monday, 16 May 1988, 8:42 PM	

Notes:

- Lowercase characters must be used only for alphabetic picture terms.
- Lowercase characters are treated as uppercase characters when the national language in effect is uppercase U.S. English.

Table 35. Japanese Eras Used by Date/Time Services when <JJJJ> 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)

Table 36. Chinese Eras Used by Date/Time Services when <CCCC> or <CCCCCCCC> Specified

First date of Chinese Era	Era Name	Era Name in Traditional Chinese DBCS Code	Valid Year (YY, ZYY) Values
1912-01-01	*MinKow*	X'0E4D8256CE0F'	01-999 (77 equals 1988)
	ChuHwaMinKow	X'0E4C845ADD4D8256CE0F'	

Appendix C. Controlling Storage Allocation

This appendix provides information about storage report statistics to help you control storage allocation.

Controlling Storage Allocation

The following run-time options control storage allocation: STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP. Ensure that these options are tuned appropriately to avoid performance problems.

To generate a report of the storage a routine (or more specifically, an enclave) used during its run, specify the RPTSTG(ON) run-time 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 run-time options for future runs. The output is written to the LE/VSE message file.

Neither the storage report nor the corresponding run-time options include the storage that LE/VSE acquires during early initialization, before run-time options processing, and before the start of space management monitoring.

Figure 13 shows a sample storage report, and the sections that follow describe the contents of the report.

```
Storage Report for Enclave CBLDATE 11/09/01 10:37:28 AM
Language Environment for VSE/ESA V1 R4.2

    STACK statistics:
      Initial size:                                    131072
      Increment size:                                  131072
      Total stack storage used (sugg. initial size):     8920
      Number of segments allocated:                         1
      Number of segments freed:                             0
    LIBSTACK statistics:
      Initial size:                                    12288
      Increment size:                                   4096
      Total stack storage used (sugg. initial size):    1152
      Number of segments allocated:                         1
      Number of segments freed:                             0
    HEAP statistics:
      Initial size:                                    32768
      Increment size:                                  32768
      Total heap storage used (sugg. initial size):      752
      Successful Get Heap requests:                         1
      Successful Free Heap requests:                        0
      Number of segments allocated:                         1
      Number of segments freed:                             0
    ANYHEAP statistics:
      Initial size:                                    16384
      Increment size:                                   8192
      Total heap storage used (sugg. initial size):     2952
      Successful Get Heap requests:                        13
      Successful Free Heap requests:                        0
      Number of segments allocated:                         1
      Number of segments freed:                             0
    BELOWHEAP statistics:
      Initial size:                                     8192
      Increment size:                                   4096
      Total heap storage used (sugg. initial size):     1184
      Successful Get Heap requests:                         6
      Successful Free Heap requests:                        3
      Number of segments allocated:                         1
      Number of segments freed:                             0
    Additional Heap statistics:
      Successful Create Heap requests:                      0
      Successful Discard Heap requests:                     0
      Total heap storage used:                              0
      Successful Get Heap requests:                         0
      Successful Free Heap requests:                        0
      Number of segments allocated:                         0
      Number of segments freed:                             0
End of Storage Report
```

Figure 13. Storage Report Produced by LE/VSE Run-Time Option RPTSTG(ON)

The statistics for initial and incremental allocations of storage types that have a corresponding run-time option differ from the run-time option settings when their values have been rounded up by the implementation, or when allocations larger than the amounts specified were required during execution. All of the following are rounded up to an integral number of double-words:

- Initial STACK allocations
- Initial allocations of all types of heap
- Incremental allocations of all types of stack and heap

Stack Storage Statistics

LE/VSE stack storage is managed at the thread level—each thread has its own stack-type resources.

Note: LE/VSE Version 1 Release 4 supports only one thread within an enclave.

STACK and LIBSTACK Statistics

Table 37. STACK and LIBSTACK Statistics

Statistic	Description
Initial size	The actual size of the initial segment assigned to each thread.
Increment size	The size of each incremental segment acquired, as determined by the increment portion of the corresponding run-time option.
Total stack storage used	The largest amount used by the thread at any one time.
Number of segments allocated	The number of incremental segments allocated.
Number of segments freed	The number of incremental segments freed.

Note: The number of incremental segments freed may be less than the number allocated if any of the segments were not freed individually during the life of the thread, but rather were freed implicitly in the course of thread termination.

Allocating Stack Storage

Another type of stack, called the reserve stack, is allocated for each thread and used to handle out-of-storage conditions. Its size is controlled by the 4th subparameter of the STORAGE run-time option, but its usage is neither tracked nor reported in the storage report.

LE/VSE acquires some initial storage, which is neither stack nor heap, at thread creation time; this storage is allocated from BELOWHEAP if ALL31(OFF) is in effect, or from ANYHEAP if ALL31(ON) is in effect.

Heap Storage Statistics

LE/VSE heap storage, is managed at the enclave level—each enclave has its own heap-type resources, which are shared by the threads that execute within the enclave.

HEAP, ANYHEAP, and BELOWHEAP Statistics

Table 38. HEAP, ANYHEAP, and BELOWHEAP Statistics

Statistic	Description
Initial size	The default initial allocation, as specified by the corresponding run-time option.
Increment size	The minimum incremental allocation, as specified by the corresponding run-time option.

HEAP, ANYHEAP, BELOWHEAP, and Additional Heap Statistics

Table 39. HEAP, ANYHEAP, BELOWHEAP, and Additional Heap Statistics

Statistic	Description
Total heap storage used	The largest total amount used by the enclave at any one time.
Successful Get Heap requests	The number of Get Heap requests.
Successful Free Heap requests	The number of Free Heap requests.
Number of segments allocated	The number of incremental segments allocated.
Number of segments freed	The number of incremental segments individually freed.

Note: The number of Free Heap requests may be less than the number of Get Heap requests if the pieces of heap storage acquired by individual Get Heap requests were not freed individually, but rather were freed implicitly in the course of enclave termination.

The number of incremental segments individually freed may be less than the number allocated if the segments were not freed individually, but rather were freed implicitly in the course of enclave termination.

These statistics, in all cases, specify totals for the whole enclave.

Additional Heap Statistics

Besides the fixed types of heap, additional types of heap can be created, each with its own heap id. You can create and discard these additional types

of heap by using the CEECRHP (see "CEECRHP—Create New Additional Heap" on page 103) and CEEDSHP (see "CEEDSHP—Discard Heap" on page 119) callable services.

Table 40. Additional Heap Statistics

Statistic	Description
Successful Create Heap requests	The number of successful Create Heap requests.
Successful Discard Heap requests	The number of successful Discard Heap requests.

Note: The number of Discard Heap requests may be less than the number of Create Heap requests if the special heaps allocated by individual Create Heap requests were not freed individually, but rather were freed implicitly in the course of enclave termination.

For more information about stack and heap storage, see *LE/VSE Programming Guide*

Language Environment Glossary

A

abend. Abnormal end of application.

absolute value. The magnitude of a real number regardless of its algebraic sign.

active routine. The currently executing routine.

additional heap. An LE/VSE heap created and controlled by a call to CEECRHP. See also *below heap*, *anywhere heap*, and *initial heap*.

addressing mode. An attribute that refers to the address length that a routine is prepared to handle upon entry. Addresses may be 24 or 31 bits long.

aggregate. A structured collection of data items that form a single data type. Contrast with *scalar*.

American National Standard Code for Information Interchange (ASCII). The code developed by the American National Standards Institute (ANSI) for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

AMODE. Addressing mode.

anywhere heap. The LE/VSE heap controlled by the ANYHEAP run-time option. It contains library data, such as LE/VSE control blocks and data structures not normally accessible from user code. The anywhere heap may reside above 16MB. See also *below heap*, *additional heap*, and *initial heap*.

application. A collection of one or more routines cooperating to achieve particular objectives.

application program. A collection of software components used to perform specific types of work on a computer, such as a program that does inventory control or payroll.

argument. An expression used at the point of a call to specify a data item or aggregate to be passed to the called routine.

array. An aggregate that consists of data objects, each of which may be uniquely referenced by subscripting.

array element. A data item in an array.

ASCII. American National Standard Code for Information Interchange.

Asian date format. In this book, Asian date format refers to the era picture strings associated with the Japanese eras or Chinese era. Era picture strings begin with a less than character < and end with a greater than character >. The characters inside are either capital Js or Cs.

assembler. see *High Level Assembler*.

automatic call library. Contains object modules that are to be used as secondary input to the linkage editor to resolve external symbols left undefined after all the primary input has been processed.

The automatic call library may be:

- Sublibraries containing object modules, with or without linkage editor control statements
- The sublibrary containing LE/VSE run-time routines (PRD2.SCEEBASE or PRD2.SCEECICS)

automatic data. Data that does not persist across calls to other routines. Automatic data may be automatically initialized to a certain value upon entry and reentry to a routine.

automatic storage. Storage that is allocated on entry to a routine or block and is freed on the subsequent return. Sometimes referred to as *stack storage* or *dynamic storage*.

B

batch. Pertaining to activity involving little or no user action. Contrast with *interactive*.

below heap. The LE/VSE heap controlled by the BELOWHEAP run-time option, which contains library data, such as LE/VSE control block and data structures not normally accessible from user code. Below heap always resides below 16MB. See also *anywhere heap*, *initial heap*, and *additional heap*.

buffer. An area of storage into which data is read or from which it is written. Typically, buffers are used only for temporary storage.

by content. See *pass by content*.

by reference. See *pass by reference*.

by value. See *pass by value*.

C

callable services. A set of services that can be invoked by an LE/VSE-conforming high level language using

the conventional LE/VSE-defined call interface, and usable by all programs sharing the LE/VSE conventions.

Use of these services helps to decrease an application's dependence on the specific form and content of the services delivered by any single operating system.

callable service stub. Contains addressing code to access LE/VSE callable service routines.

callee. Receiver of a call.

caller. A routine that calls another routine.

century window. The 100-year interval in which LE/VSE assumes all 2-digit years lie. The LE/VSE default century window begins 80 years before the system date.

chained list. Synonym for *linked list*.

child enclave. The *nested enclave* created as a result of certain commands being issued from a *parent enclave*.

CICS. Customer Information Control System.

CICS run unit. Consists of a statically and/or dynamically bound set of one or more phases which can be loaded by a CICS loader. A CICS run unit is equivalent to an LE/VSE *enclave*.

CICS translator. A routine that accepts as input an application containing EXEC CICS commands and produces as output an equivalent application in which each CICS command has been translated into the language of the source.

COBOL. COmmon Business-Oriented Language. A high level language, based on English, that is primarily used for business applications.

COBOL run unit. A COBOL-specific term that defines the scope of language semantics. Equivalent to an LE/VSE *enclave*.

command line. The command used to invoke an application program, and the associated program arguments and LE/VSE run-time options. This can be the job control EXEC statement and the associated PARM parameter, or the parameter string passed to the C `system()` function.

COMMAREA. A communication area made available to applications running under CICS.

compilation unit. An independently compilable sequence of HLL statements. Each HLL product has different rules for what makes up a compilation unit. Synonym for *program unit*.

condition. An exception that has been enabled, or recognized, by LE/VSE and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and result in an interrupt. They can also be detected by language-specific generated code or language library code.

condition handler. A user-written condition handler or language-specific condition handler (such as a PL/I ON-unit) invoked by the LE/VSE *condition manager* to respond to conditions.

condition handling. In LE/VSE, the diagnosis, reporting, and/or tolerating of errors that occur in the run-time environment.

condition manager. Manages conditions in the common execution environment by invoking various user-written and language-specific *condition handlers*.

condition step. The step of the LE/VSE condition handling model that follows the enablement step. In the condition step, user-written condition handlers and PL/I ON-units are first given a chance to handle a condition. See also *enablement step* and *termination imminent step*.

condition token. In LE/VSE, a data type consisting of 96 bits (12 bytes). The condition token contains structured fields that indicate various aspects of a condition including the severity, the associated message number, and information that is specific to a given instance of the condition.

constructed reentrancy. The attribute of applications that contain external data and require additional processing to make them reentrant. Contrast with *natural reentrancy*.

Customer Information Control System (CICS). CICS is an OnLine Transaction Processing (OLTP) system that provides specialized interfaces to databases, files and terminals in support of business and commercial applications.

D

data type. The properties and internal representation that characterize data.

DBCS. Double-byte character set.

decimal overflow. A condition that occurs when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the result.

default. A value that is used when no alternative is specified.

direct parameter passing. Placing a value directly in the parameter list body.

disabled/enabled. See *enabled/disabled*.

double-byte character set (DBCS). A collection of characters represented by a two-byte code.

double-precision. Pertaining to the use of two computer words to represent a number in accordance with the required precision. See also *precision* and *single-precision*.

doubleword. A sequence of bits or characters that comprises two computer words and can be addressed as a unit.

DSA. Dynamic storage area.

dynamic call. A call that results in the resolution of the called routine at run time. Contrast with *static call*.

dynamic storage. Storage acquired as needed at run time. Contrast with *static storage*.

dynamic storage area (DSA). An area of storage obtained during the running of an application that consists of a register save area and an area for automatic data, such as program variables. DSAs are generally allocated within LE/VSE-managed stack segments. DSAs are added to the stack when a routine is entered and removed upon exit in a last in, first out (LIFO) manner. In LE/VSE, a DSA is known as a *stack frame*.

E

EBCDIC. Extended binary-coded decimal interchange code.

EIB. EXEC interface block.

enabled/disabled. A condition is enabled when its occurrence will result in the execution of condition handlers or in the performance of a standard system action to handle the condition as defined by LE/VSE.

A condition is disabled when its occurrence will apparently be ignored by the condition manager.

enablement. The determination by a language at run time that an exception should be processed as a condition. This is the capability to intercept an exception and to determine whether it should be ignored or not; unrecognized exceptions are always defined to be enabled. Normally, enablement is used to supplement the hardware for capabilities that it does not have and for language enforcement of the language's semantics. An example of supplementing the hardware is the specialized handling of floating-point overflow exceptions based on language specifications (on some machines this can be achieved through masking the exception).

enablement step. The first step of the LE/VSE condition handling model. In the enablement step it is determined whether an exception is to be *enabled* and processed as a condition. See also *condition step* and *termination imminent step*.

enclave. In LE/VSE, an independent collection of routines, one of which is designated as the main routine. An enclave is roughly analogous to a program or run unit.

entry name. In assembler language, a programmer-specified name within a control section that identifies an entry point and can be referred to by any control section.

entry point. In assembler language, the address or label of the first instruction that is executed when a routine is entered for execution.

environment. A set of services and data available to a program during execution. In LE/VSE, environment is normally a reference to the run-time environment of HLLs at the enclave level.

epilog. Code generated at the end of a routine, normally causing a return to the caller of the routine.

ESDS. Entry sequenced data sets. See *VSAM*.

EXEC interface block (EIB). In CICS, a control block containing information useful in the execution of an application, such as a transaction identifier and a time and a date when the transaction is started.

execution time. Synonym for *run time*.

execution environment. Synonym for *run-time environment*.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

external data. Data that persists over the lifetime of an enclave and maintains last-used values whenever a routine within the enclave is reentered. Within an enclave consisting of a single phase, it is equivalent to COBOL external data.

external routine. A procedure or function that may be invoked from outside the program in which the routine is defined.

F

facility ID. A string of three characters identifying an LE/VSE-conforming component. The facility IDs assigned by IBM are:

CEE LE/VSE common library
EDC C language-specific library
IGZ COBOL language-specific library
IBM PL/I language-specific library

feedback code (fc). A condition token value. If you specify *fc* in a call to a callable service, a condition

token indicating whether the service completed successfully is returned to the calling routine.

file. A named collection of related data records that is stored and retrieved by an assigned name.

filename. A 1- to 7-character name used within an application and in JCL to identify a file. The filename provides the means for the logical file to be connected to the physical file.

fix-up and resume. The correction of a condition by changing the argument or parameter and running the routine again.

fixed-overflow. A condition raised as a result of an overflow during signed binary arithmetic or signed left-shift operations.

function. A routine that is invoked by coding its name in an expression. The routine passes a result back to the invoker through the routine name.

G

Gregorian calendar. The calendar in use since Friday, 15 October, 1582 throughout most of the world. Used as the basis for the *Lilian date* used in many LE/VSE date and time services.

H

handle cursor. Points to the first condition handler within the stack frame that is to be invoked when a condition occurs. As condition handling progresses, the handle cursor moves to earlier handlers within the stack frame, or to the first handler in the calling stack frame.

header file. A file that contains system-defined control information that precedes user data.

heap 0. Synonym for *initial heap*.

heap. An area of storage used for allocation of storage whose lifetime is not related to the execution of the current routine. The heap consists of the initial heap segment and zero or more increments. See also *additional heap*, *anywhere heap*, *below heap*, *heap element*, and *initial heap*.

heap element. A contiguous area of storage allocated by a call to the CEEGTST service. Heap elements are always allocated within a single heap segment.

heap increment. See *increment*.

heap segment. A contiguous area of storage obtained directly from the operating system. The LE/VSE storage management scheme subdivides heap segments into individual heap elements. If the initial heap segment becomes full, LE/VSE obtains a second segment, or increment, from the operating system.

heap storage. See *heap*.

hexadecimal. A base 16 numbering system. Hexadecimal digits range from 0 through 9 (decimal 0 to nine) and uppercase or lowercase A through F (decimal ten to fifteen).

High Level Assembler. An IBM licensed program. Translates symbolic assembler language into binary machine language.

high level language (HLL). A programming language above the level of assembler language and below that of program generators and query languages.

HLL. High level language.

I

ILC. Interlanguage communication.

increment. The second and subsequent segments of storage allocated to the stack or heap.

indirect argument passing. The body of the argument list contains a pointer to the argument value.

indirect parameter passing. Placing an address in a parameter list. In other words, passing a pointer to a value instead of passing the value itself.

initial heap. The LE/VSE heap controlled by the HEAP run-time option and designated by a *heap_id* of 0. The initial heap contains dynamically allocated user data. See also *additional heap*.

initial heap segment. The first heap segment. A heap consists of the initial heap segment and zero or more additional segments or increments.

initial stack segment. The first stack segment. A stack consists of the initial stack segment and zero or more additional segments or increments.

initial program load (IPL). The process of loading system programs and preparing a system to run jobs.

IPL. Initial program load.

input procedure. A set of statements, to which control is given during the execution of a SORT statement, for the purpose of controlling the release of specified records to be sorted.

instance specific information (ISI). Located within the LE/VSE condition token, the ISI contains information used by the condition manager to identify and react to a specific occurrence of a condition.

integer. A positive or negative whole number or zero.

interactive. Pertaining to a program or system that alternately accepts input and responds. In an interactive system, a constant dialog exists between user and system. Contrast with *batch*.

interlanguage communication (ILC). The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages.

interrupt. A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

ISI. Instance specific information.

J

JCL. Job control language.

job control language (JCL). A sequence of commands used to identify a job to an operating system and to describe a job's requirements.

job step. One of a group of related programs complete with the JCL statements necessary for a particular run. Every job step is identified in the job stream by an EXEC statement under one job statement for the whole job.

Julian date. A date format that contains the year in positions 1 and 2, and the day in positions 3 through 5. The day is represented as 1 through 366, right-adjusted, with zeros in the unused high-order position.

K

KSDS. Key sequenced data sets. See *VSAM*.

L

Language Environment. A set of architectural constructs and interfaces that provides a common run-time environment and run-time services to applications compiled by Language Environment-conforming compilers.

Language Environment for VSE/ESA. An IBM software product that is the implementation of Language Environment on the VSE platform.

LE/VSE. Short form of Language Environment for VSE/ESA.

LE/VSE-conforming. Adhering to LE/VSE's common interface.

library. A collection of functions, subroutines, or other data.

library stack. An independent area of stack storage, allocated below the 16MB line, designed to be used only by library routines. See also *stack*, *user stack*, and *stack frame*.

library vector table (LIBVEC). A vector table used to support access to library routines (LE/VSE and HLLs) from compiler-generated code, user-written assembly language code, and other subroutines.

LIBVEC. Library vector table

LIFO. Last in, first out method of access. A queuing technique in which the next item to be retrieved is the item most recently placed in the queue.

Lilian date. The number of days since the beginning of the Gregorian calendar. Day one is Friday, 15 October 1582. The Lilian date format is named in honor of Luigi Lilio, the creator of the Gregorian calendar.

linked list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for *chained list*.

linkage editor. A program that resolves cross-references between separately assembled object modules and then assigns final addresses to create a single relocatable phase. The linkage editor then stores the phase in a program library in main storage.

link-edit. To create a loadable computer program by means of a linkage editor.

local data. Data that is known only to the routine in which it is declared. Equivalent to local data in C and WORKING-STORAGE in COBOL.

locale. The definition of the subset of a user's environment that depends on language and cultural conventions.

locator. PL/I control block that holds the address of data such as structures or arrays and the address of the descriptor.

LWS. Library workspace.

M

main program. The first routine in an enclave to gain control from the invoker.

mapped condition. A condition that is generated by one component and converted, or mapped, to another component; for example, some LE/VSE conditions, such as the decimal divide condition that maps directly to the PL/I ZERODIVIDE condition.

megabyte (M). 1,048,576 bytes.

module. A language construct that consists of procedures or data declarations and can interact with other such constructs. In PL/I, an external procedure.

multitasking. See *multithreading*.

multithreading. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks, or threads.

N

n-way ILC application. An ILC application that includes a C routine, COBOL program, and PL/I routine.

NAB. Next available byte.

name scope. The portion of an application within which a particular declaration of external data applies or is known.

name space. The portion of a phase within which a particular declaration of external data applies or is known.

named heap. A heap set up specifically by the CEECRHP callable service. An identifier is returned when the heap is created.

national language support. Translation requirements affecting parts of licensed programs; for example, translation of message text and conversion of symbols specific to countries.

natural reentrancy. The attribute of applications that contain no static external data and do not require additional processing to make them reentrant. Contrast with *constructed reentrancy*.

nested condition. A condition that occurs during the handling of another, previous condition. LE/VSE by default permits 10 levels of nested conditions. You may change this setting by altering the DEPTHCONDLMT run-time option.

nested program. In COBOL, a program that is directly contained within another program.

nested enclave. A new enclave created by an existing enclave. The nested enclave that is created must be a new main routine within the process. See also *child enclave* and *parent enclave*.

next available byte (NAB). The address of the next available byte of storage on a doubleword boundary. This address is a segment of stack storage.

next sequential instruction. The next instruction to be executed in the absence of any branch or transfer of control.

non-LE/VSE conforming. Any HLL program that does not adhere to LE/VSE's common interface. For example, VS COBOL II, DOS/VS COBOL, and DOS/VS PL/I are all non-LE/VSE conforming HLLs. Synonym for *pre-LE/VSE conforming*.

non-reentrant. A type of program that cannot be shared by multiple users.

O

object code. Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code.

object deck. Synonym for *object module*.

object module. A portion of an object program suitable as input to a linkage editor. Synonym for *object deck*.

offset. The number of measuring units from an arbitrary starting point in a record, area, or control block, to some other point.

omitted parameter. A parameter not needed in a call.

online. Pertaining to a user's ability to interact with a computer.

ON-unit. The specified action to be taken upon detection of the condition named in the containing ON statement.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

out-of-storage condition. A condition signaled when an application has used all of the storage allocated to it. If the STORAGE run-time option is set to a value other than 0, LE/VSE adds a reserve stack segment to the overflowing stack, and then signals the out-of-storage condition.

output procedure. A set of statements, to which control is given during the execution of a SORT statement after the sort function is completed, or during the MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

overflow. That portion of an operation that exceeds the capacity of the intended unit of storage.

overlay. To write over existing data in storage.

owning stack frame. Given the calling sequence of Routine 1 calling Routine 2 that in turn calls Routine 3, Routine 3 is the owning stack frame if a condition occurs while Routine 3 is executing.

ON-unit. The specified action to be taken upon detection of the condition named in the containing ON statement.

P

packed decimal format. A format in which each byte in a field except the rightmost byte represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1100.

pad. To fill unused positions in a field with dummy data, usually zeros, ones, or blanks.

parameter. Data items that are received by a routine.

parent enclave. The enclave that issues a call to system services or language constructs to create a nested (child) enclave. See also *child enclave* and *nested enclave*.

partition. A fixed-size division of storage.

Pascal. A high level language for general purpose use. Programs written in Pascal are block structured, consisting of independent routines.

pass by content. A COBOL argument passing style synonymous with passing an argument by value directly. In this style, R1 contains a pointer to a copy of the argument.

pass by reference. In programming languages, one of the basic argument passing semantics. The address of the object is passed. Any changes made by the callee to the argument value will be reflected in the calling routine at the time the change is made.

pass by value. In programming languages, one of the basic argument passing semantics. The value of the object is passed. Any changes made by the callee to the argument value will not be reflected in the calling routine.

percolate. The action taken by the condition manager when the returned value from a condition handler indicates that the handler could not handle the condition, and the condition will be transferred to the next handler.

phase. An application or routine in a form suitable for execution. The application or routine has been compiled and link-edited; that is, address constants have been resolved.

picture string. Character strings used to specify date and time formats.

PL/I. A general purpose scientific/business high level language. It is a high-powered procedure-oriented language especially well suited for solving complex scientific problems or running lengthy and complicated business transactions and record-keeping applications.

pointer. A data element that indicates the location of another data element.

portability. The ability to transfer an application from one platform to another with relatively few changes to the source code.

PPA1 entry point block. Program Prolog Area. This block contains information about the compiled module.

PPA2 entry point block. An extension of the PPA1 entry point block.

PPT. Processing Program Table

precedence. In programming languages, an order relation defining the sequence of the application of operations or options.

precision. A measure of the ability to distinguish between nearly equal values. See also *single-precision* and *double-precision*.

pre-initialization. A facility that allows a routine to initialize the run-time environment once, perform multiple executions within the environment, then explicitly terminate the environment.

pre-LE/VSE conforming. Any HLL program that does not adhere to LE/VSE's common interface. For example, VS COBOL II, DOS/VS COBOL, and DOS/VS PL/I are all pre-LE/VSE conforming HLLs. Synonym for *non-LE/VSE conforming*.

preprocessor. A routine that examines application source code for preprocessor statements that are then executed, resulting in the alteration of the source.

procedure. A named block of code that can be invoked, usually via a call. In LE/VSE, the term *routine* is used as generic for a procedure or a function.

process. The highest level of the LE/VSE program management model. A process is a collection of resources, both program code and data, and consists of at least one enclave.

Processing Program Table (PPT). A CICS table that contains information about CICS phases (whether the phase is in storage or not, its language, use count and entry point address, etc.) needed to complete a transaction.

program. See *application program*.

program control data. In PL/I, data used to affect how a program runs; that is, any data that is not string or arithmetic data.

program management. The functions within the system that provide for establishing the necessary

activation and invocation for a program to run in the applicable run-time environment when it is called.

program mask. A structure that describes the manner in which S/370 hardware-detected conditions are to be handled.

program status word (PSW). An area in storage used to indicate the order in which instructions are executed and to hold and indicate the status of the operating system. The program mask (bits 20 to 23) of the PSW can be manipulated to enable or disable the detection of some hardware conditions under LE/VSE.

program unit. Synonym for *compilation unit*.

programmable workstation (PWS). A workstation that has some degree of processing capability and that allows a user to change its functions.

program unit. Synonym for *compilation unit*.

prolog. The code sequence when a routine is entered.

promote. To change a condition. A condition is promoted when a condition handling routine changes the condition to a different one. A condition handling routine promotes a condition because the error needs to be handled in a way other than that suggested by the original condition.

PSB. Program specification block.

PSW. Program status word.

Q

q_data. Qualifying data. Information that a user-written condition handler can use to identify and react to a given instance of a condition.

q_data_token. An optional 32-bit data object that is placed in the ISI. It is used to access math condition qualifying data associated with a given instance of a condition.

R

reason code. A value returned to the invoker of an enclave that indicates how the enclave terminated. The value reflects whether the enclave terminated successfully, or unsuccessfully, to an unhandled condition.

recursive routine. A routine that can call itself or be called by another routine that it has called.

reentrant. The attribute of a routine or application that allows more than one user to share a single copy of a phase.

register. (1) Special processing areas that hold a specific amount of data and can process, load, and store this data quickly. (2) To specify formally. In LE/VSE, to register a condition handler means to add a user-written condition handler onto a routine's stack frame.

register save area (RSA). Area of main storage in which contents of registers are saved.

resident routines. A category of LE/VSE library routines linked with your application. They include such things as initialization routines and *callable service stubs*.

resume. To begin execution in an application at the point immediately after which a condition occurred. A resume occurs when the condition manager determines that a condition has been handled and normal application execution should continue.

resume cursor. Designates the point in the application where a condition occurred when it is first reported to the condition manager. The resume cursor also designates the point where execution resumes after a condition is handled, usually at the instruction in the application immediately following the point at which the error occurred. The resume cursor can be moved with the CEEMRCR callable service.

return code. A code produced by a routine to indicate its success. It may be used to influence the execution of succeeding instructions.

return_code_modifier. A value set by LE/VSE routines that manage the environment. It indicates whether or not an enclave terminated successfully.

rollback. The backing out of any updates made by a failing application.

root phase. The phase containing a main routine and the first to be executed in an application.

routine. In this book, used as an exact equivalent of a COBOL/VSE *compilation unit*, and means a named external routine, with or without named entry points, and with or without internal (contained) routines.

RMODE. Residence mode. The attribute of a phase that specifies whether the phase, when loaded, must reside below the 16MB virtual storage line or may reside anywhere in virtual storage.

RRDS. Relative record data sets. See *VSAM*.

run. To cause a program, utility, or other machine function to be performed.

run time. Any instant at which a program is being executed. Synonym for *execution time*.

run-time environment. A set of resources that are used to support the execution of a program. Synonym for *execution environment*.

run unit. One or more object programs that are executed together. In LE/VSE, a run unit is the equivalent of an *enclave*.

S

safe condition. Any condition having a severity of 0 or 1. Such conditions are ignored if no condition handler handles the condition.

save area. Area of main storage in which contents of registers are saved.

scalar. A quantity characterized by a single value. Contrast with *aggregate*.

scope. 1. A term used to describe the effective range of the enablement of a condition and/or the establishment of a user-generated routine to handle a condition. Scope can be both statically and dynamically defined. 2. The portion of an application within which the definition of a variable remains unchanged.

segment. See *stack segment*.

severity code. A part of run-time messages that indicates the severity of the error condition (1, 2, 3, or 4).

shared virtual area (SVA). In VSE, an area of main storage containing a system directory list (SDL) of frequently used phases, resident routines shared between partitions, and an area for system support. The presence of a reentrable routine in the SVA saves loading time when the routine is needed.

significance condition. A condition raised when the resulting fraction in a floating-point addition or subtraction operation is zero.

single-precision. Pertaining to the use of one computer word to represent a number in accordance with the required precision. Needed for proper alignment. See also *precision* and *double-precision*.

sort/merge program. A processing program that can be used to sort or merge records in a prescribed sequence.

source code. The input to a compiler or assembler, written in a source language.

source program. A set of instructions written in a programming language that must be translated to machine language before the program can be run.

stack. An area of storage used for suballocation of stack frames. Such suballocations are allocated and freed on a LIFO (last in, first out) basis. A stack is a collection of one or more stack segments consisting of an initial stack segment and zero or more increments.

stack frame. The physical representation of the activation of a routine. The stack frame is allocated on a LIFO stack and contains various pieces of information including a save area, condition handling routines, fields to assist the acquisition of a stack frame from the stack, and the local, automatic variables for the routine. In LE/VSE, a stack frame is synonymous with *DSA*.

stack frame collapse. An action that occurs when the condition manager skips over one or more active routines and execution resumes in an earlier routine on the stack. A stack frame collapse happens if an explicit GOTO is coded in a PL/I routine or if the resume cursor is moved with the CEEMRCR.

stack segment. A contiguous area of storage obtained directly from the operating system. The LE/VSE storage management scheme subdivides stack segments into individual DSAs. If the initial stack segment becomes full, a second segment or increment is obtained from the operating system.

stack storage. See *stack* and *automatic storage*.

standard system action. The name given to the language-defined default action taken when a condition occurs and it is not handled by a condition handler.

statement. In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations.

static call. A call that results in the resolution of the called program statically at link-edit time. Contrast with *dynamic call*.

static data. Data that retains its last-used state across calls.

static storage. Storage that persists and retains its value across calls. Contrast with *dynamic storage*.

storage heap. An unordered group of program stack areas that may be associated with programs running within a process.

suboption. An option that can be used with compile-time and run-time options to further specify the action of the option.

subroutine. In general, any routine within an application called by another routine.

symbolic feedback code. The symbolic representation of the 12-byte condition token returned by LE/VSE callable services. Symbolic feedback codes are provided so that you do not have to code the entire 12-byte condition token in a condition handling routine.

system directory list (SDL). In VSE, a list containing the directory entries of frequently-used phases and of all phases resident in the SVA. The list resides in the SVA.

subsystem. A secondary or subordinate system, or programming support, usually capable of operating independently of or asynchronously with a controlling system. Example: CICS.

SVC. Supervisor call. A request that serves as the interface to certain functions, such as the allocation of storage.

syntax. The rules governing the structure of a programming language and the construction of a statement in a programming language.

T

thread. The basic run-time path within the LE/VSE program management model. It is dispatched by the system with its own instruction counter and registers. The thread is where actual code resides.

token. See *condition token*.

translator. See *CICS translator*.

transient data queue. A file to which run-time messages are written under CICS. Under LE/VSE, the name of this file is CESE.

termination imminent step. The final step of the 3-step LE/VSE condition handling model. In the termination imminent step, user-written condition handlers and PL/I ON-units are given one last chance to handle a condition or perform cleanup before the thread is terminated. See also *condition step* and *enablement step*.

U

underflow condition. A condition that occurs when the result characteristic of a floating-point operation is less that zero and the result fraction is not zero. In an extended-format floating-point result, the condition is raised only when the high-order characteristic overflows.

unpacked decimal format. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte. Bits 0 through 3 of all other bytes contain 1s (hex F). For example, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Synonym for *zoned decimal format*.

user-written condition handler. A routine established by the CEEHDLR callable service to handle a condition or conditions when they occur in the common run-time environment. A queue of user-written condition handlers established by CEEHDLR may be associated with each stack frame in which they are established.

user exit. A routine that takes control at a specific point in an application. Two assembler user exits and one HLL user exit are provided by LE/VSE. They are invoked to perform initialization functions and both normal and abnormal termination functions.

user heap. See *initial heap*.

user stack. An independent area of stack storage that may be located above or below 16MB, designed to be used by both library routines and compiled code. See also *stack*, *stack frame*, *library stack*.

V

vendor. A person or company that provides a service or product to another person or company.

VSTRING. The VSTRING data type is used for the character string parameters in many of the LE/VSE callable services. In Language Environment/VSE Version 1 Release 4, VSTRING is a halfword length-prefixed character string for input, or a fixed-length 80-character string for output.

VSE (Virtual Storage Extended). A system that consists of a basic operating system (VSE/Advanced Functions) and IBM-supplied programs required to meet the data processing needs of a user.

W

weak external reference. A special type of external reference that is not to be resolved by automatic library calls unless an ordinary external reference to the same symbol is found. The external symbol dictionary entry specifies the symbol; the location is unknown.

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

working storage. In COBOL/VSE, the storage required for data items in the WORKING-STORAGE SECTION. Working storage is a portion of main storage that is used by a computer program to hold data temporarily.

Z

zoned decimal format. Synonym for *unpacked decimal format*.

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

N

O

P

Q

R

S

T

U

V

W

X

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

IBM Language Environment for VSE/ESA
Programming Reference
Version 1 Release 4 Modification Level 4

Publication No. SC33-6685-05

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

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

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

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

Please tell us how we can improve this book:

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

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

Name _____ Address _____

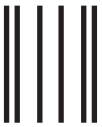
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC33-6685-05

Cut or Fold
Along Line

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



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

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

Cut or Fold
Along Line

File Number: S370/S390-40
Program Number: 5686-CF7

Printed in USA

SC33-6685-05

Spine information:

LE/VSE

Programming Reference

Version 1
Release 4 Modification
Level 4

SC33-6685-05