Language Environment Programming Reference
**Figures**

1. Example syntax for the CEEOPTS DD statement 8
2. Effect of DEPTHCONDLMT(3) on Condition Handling ............. 25
3. An Invalid COBOL CALL that Omits the fc Parameter. ............. 112
4. An Invalid PL/I CALL that Omits the fc Parameter. ............. 112
5. Valid COBOL CALLs that Use the Optional fc Parameter ............. 112
6. Valid PL/I CALLs that Use the Optional fc Parameter ............. 112
7. Sample Callable Services Invocation Syntax for C/C++ ............. 115
8. Sample Callable Services Invocation Syntax for COBOL .......... 116
9. Sample Callable Services Invocation Syntax for PL/I ............ 117
10. C/C++ Example of CEE3ABD ........... 123
11. COBOL Example of CEE3ABD ........... 123
12. PL/I Example of CEE3ABD ........... 124
13. C/C++ Example of CEE3AB2 .......... 126
14. COBOL Example of CEE3AB2 .......... 127
15. PL/I Example of CEE3AB2 .......... 128
16. C/C++ Example of CEE3CIB .......... 130
17. COBOL Example of CEE3CIB .......... 131
18. PL/I Example of CEE3CIB .......... 132
19. C/C++ Example of CEE3CTY .......... 135
20. COBOL Example of CEE3CTY .......... 136
21. PL/I Example of CEE3CTY .......... 137
22. C/C++ Example of CEE3DLY .......... 139
23. COBOL Example of CEE3DLY .......... 139
24. PL/I Example of CEE3DLY .......... 141
25. C/C++ Example of CEE3DMP .......... 147
26. COBOL Example of CEE3DMP .......... 148
27. PL/I Example of CEE3DMP .......... 149
28. C/C++ main() Routine that Calls CEEHDLR and CEE3GRC .......... 151
29. C/C++ User-Written Condition Handler that Sets a User Enclave Return Code .......... 152
30. C/C++ Subroutine that Generates the Divide-by-Zero Condition .......... 152
31. COBOL Main Routine that Calls CEEHDLR and CEE3GRC .......... 153
32. COBOL Condition Handler that Sets a User Enclave Return Code and Resumes when a Divide-by-Zero Condition Occurs .......... 154
33. COBOL Subroutine that Generates a Divide-by-Zero .......... 157
34. PL/I Example that Sets and Retrieves the User Enclave Return Code when a Divide-by-Zero is Generated .......... 157
35. C/C++ Example of CEE3GRN .......... 159
36. COBOL Example of CEE3GRN .......... 161
37. PL/I Example of CEE3GRN .......... 164
38. COBOL Example of CEE3GRO .......... 165
39. PL/I Example of CEE3GRO .......... 168
40. C/C++ Example of CEE3INF .......... 171
41. COBOL Example of CEE3INF .......... 172
42. PL/I Example of CEE3INF .......... 173
43. C/C++ Example of CEE3LNG .......... 177
44. COBOL Example of CEE3LNG .......... 177
45. PL/I Example of CEE3LNG .......... 179
46. C/C++ Example of CEE3MCS .......... 183
47. COBOL Example of CEE3MCS .......... 184
48. PL/I Example of CEE3MCS .......... 185
49. C/C++ Example of CEE3MC2 .......... 187
50. COBOL Example of CEE3MC2 .......... 188
51. PL/I Example of CEE3MC2 .......... 189
52. C/C++ Example of CEE3MDS .......... 190
53. COBOL Example of CEE3MDS .......... 191
54. PL/I Example of CEE3MDS .......... 192
55. C/C++ Example of CEE3MTS .......... 193
56. COBOL Example of CEE3MTS .......... 194
57. PL/I Example of CEE3MTS .......... 195
58. C/C++ Example of CEE3PRM .......... 196
59. COBOL Example of CEE3PRM .......... 197
60. PL/I Example of CEE3PRM .......... 198
61. C/C++ Example of CEE3PR2 .......... 200
62. COBOL Example of CEE3PR2 .......... 201
63. PL/I Example of CEE3PR2 .......... 202
64. C/C++ Example of CEE3PRH .......... 204
65. COBOL Example of CEE3PRH .......... 205
66. PL/I Example of CEE3PRH .......... 206
67. C/C++ Example of CEE3SPM .......... 209
68. COBOL Example of CEE3SPM .......... 210
69. PL/I Example of CEE3SPM .......... 211
70. C/C++ Example of CEE3USR .......... 215
71. COBOL Example of CEE3USR .......... 216
72. PL/I Example of CEE3USR .......... 217
73. COBOL Example of CEECBLDY .......... 221
74. C/C++ Example of CEECMO .......... 224
75. COBOL Example of CEECMO .......... 225
76. PL/I Example .......... 227
77. C/C++ Example of CEECRHP .......... 230
78. COBOL Example of CEECRHP .......... 231
79. PL/I Example of CEECRHP .......... 232
80. C/C++ Example of CEECZST .......... 234
81. COBOL Example of CEECZST .......... 235
82. PL/I Example of CEECZST .......... 236
83. C/C++ Example of CEEDATE .......... 240
84. COBOL Example of CEEDATE .......... 240
85. PL/I Example of CEEDATE .......... 242
86. C/C++ Example of CEEDATM .......... 246
87. COBOL Example of CEEDATM .......... 246
88. PL/I Example of CEEDATM .......... 249
89. C/C++ Example of CEEDAYS .......... 253
90. COBOL Example of CEEDAYS .......... 254
91. PL/I Example of CEEDAYS .......... 255
92. C/C++ Example of CEEDCOC .......... 259
93. COBOL Example of CEEDCOC .......... 260
94. PL/I Example of CEEDCOC .......... 261
95. C/C++ Example of CEEDLYM .......... 264
Tables

1. How to Use z/OS Language Environment Publications ........................................... xii
2. Syntax examples ........................................................................................................... xiv
3. Run-Time Options Quick Reference - AMODE
   31 .............................................................................................................................. 3
4. Run-Time Options Quick Reference - AMODE
   64 .............................................................................................................................. 5
5. Condition Handling of 0Cx ABENDS ................................................................. 81
6. Handling of Software Raised Conditions ............................................................. 81
7. TRAP Run-Time Option Settings ........................................................................... 93
8. Bit Manipulation Routines ....................................................................................... 105
9. Condition Handling Callable Services .................................................................... 105
10. Date and Time Callable Services ............................................................................ 106
11. Dynamic Storage Callable Services ........................................................................ 106
12. General Callable Services ....................................................................................... 107
13. Initialization and Termination Services .................................................................. 107
14. Locale Callable Services .......................................................................................... 107
15. Math Services ........................................................................................................... 108
16. Message Handling Callable Services ...................................................................... 109
18. Files Used in C/C++, COBOL, and PL/I Examples .................................................. 113
19. Imbedding Files in Your Routines ............................................................................ 114
20. Data Type Definitions for C/C++ ............................................................................. 118
21. Data Type Definitions for COBOL .......................................................................... 119
22. Data Type Definitions for PL/I ............................................................................... 120
23. National Language Codes ....................................................................................... 180
25. HEAP Attributes Based on the Setting of the options Parameter ....................... 228
26. Sample Output of CEEDATE ................................................................................... 243
27. Sample Output of CEEDATM .................................................................................. 249
28. Examples of calls to CEESLOG. ............................................................................. 477
29. C/C++ examples ....................................................................................................... 510
30. COBOL examples .................................................................................................... 510
31. PL/I examples .......................................................................................................... 511
32. Defaults Currency and Picture Strings Based on COUNTRY Setting .................... 515
33. Picture Character Terms Used in Picture Strings for Date and Time Services ....... 521
34. Examples of Picture Strings Recognized by Date and Time Services ................... 522
35. Japanese Eras Used by Date/Time Services When <JJJJ> Specified ..................... 522
About this document

This document supports z/OS (5694-A01).

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

Language Environment is the prerequisite run-time environment for applications generated with the following IBM compiler products:

- z/OS XL C/C++ (feature of z/OS)
- z/OS C/C++
- OS/390 C/C++
- C/C++ for MVS/ESA
- C/C++ for z/VM
- XL C/C++ for z/VM
- AD/Cycle® C/370™
- VisualAge for Java, Enterprise Edition for OS/390
- Enterprise COBOL for z/OS
- Enterprise COBOL for z/OS and OS/390
- COBOL for OS/390 & VM
- COBOL for MVS & VM (formerly COBOL/370)
- Enterprise PL/I for z/OS
- Enterprise PL/I for z/OS and OS/390
- VisualAge PL/I
- PL/I for MVS & VM (formerly PL/I MVS™ & VM)
- VS FORTRAN and FORTRAN IV (in compatibility mode)

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

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

Debug Tool is also available as a standalone product. Debug Tool Utilities and Advanced Functions is also available. For more information, see


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

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

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

This publication provides application programmers with a detailed description of each Language Environment 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 Language Environment-conforming high-level languages (HLLs) and assembler language. Before using Language Environment, you should be familiar with the HLLs in which your applications are written. You should also understand the operating systems and any subsystems in which you plan to run Language Environment applications.

Using your documentation

The publications provided with Language Environment are designed to help you:

- Manage the run-time environment for applications generated with a Language Environment-conforming compiler.
- Write applications that use the Language Environment callable services.
- Develop interlanguage communication applications.
- Customize Language Environment.
- Debug problems in applications that run with Language Environment.
- Migrate your high-level language applications to Language Environment.

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

Each publication helps you perform different tasks, some of which are listed in Table 1. All books are available in printable (PDF) and BookManager softcopy formats. For a complete list of publications that you may need, see "Bibliography" on page 533.

Table 1. How to Use z/OS Language Environment Publications

<table>
<thead>
<tr>
<th>To ...</th>
<th>Use ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate Language Environment</td>
<td>z/OS Language Environment Concepts Guide</td>
</tr>
<tr>
<td>Plan for Language Environment</td>
<td>z/OS Language Environment Concepts Guide</td>
</tr>
<tr>
<td>Install Language Environment</td>
<td>z/OS Program Directory</td>
</tr>
<tr>
<td>Customize Language Environment</td>
<td>z/OS Language Environment Customization</td>
</tr>
<tr>
<td>Understand Language Environment program models and concepts</td>
<td>z/OS Language Environment Concepts Guide</td>
</tr>
<tr>
<td></td>
<td>z/OS Language Environment Programming Guide</td>
</tr>
<tr>
<td></td>
<td>z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode</td>
</tr>
<tr>
<td>Find syntax for Language Environment run-time options and callable services</td>
<td>z/OS Language Environment Programming Reference</td>
</tr>
<tr>
<td>Develop applications that run with Language Environment</td>
<td>z/OS Language Environment Programming Guide and your language programming guide</td>
</tr>
<tr>
<td>Debug applications that run with Language Environment, diagnose problems with Language Environment</td>
<td>z/OS Language Environment Debugging Guide</td>
</tr>
<tr>
<td>Get details on run-time messages</td>
<td>z/OS Language Environment Run-Time Messages</td>
</tr>
</tbody>
</table>
Table 1. How to Use z/OS Language Environment Publications (continued)

<table>
<thead>
<tr>
<th>To</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop interlanguage communication (ILC) applications</td>
<td><a href="#">z/OS Language Environment Writing Interlanguage Communication Applications</a> and your language programming guide</td>
</tr>
<tr>
<td>Migrate applications to Language Environment</td>
<td><a href="#">z/OS Language Environment Run-Time Application Migration Guide</a> and the migration guide for each Language Environment-enabled language</td>
</tr>
</tbody>
</table>

### How to read syntax diagrams

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

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

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

#### Symbols

The following symbols may be displayed in syntax diagrams:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>➤➤</td>
<td>Indicates the beginning of the syntax diagram.</td>
</tr>
<tr>
<td>➤➤</td>
<td>Indicates that the syntax diagram is continued to the next line.</td>
</tr>
<tr>
<td>➤➤</td>
<td>Indicates that the syntax is continued from the previous line.</td>
</tr>
<tr>
<td>➤➤</td>
<td>Indicates the end of the syntax diagram.</td>
</tr>
</tbody>
</table>

#### Syntax items

Syntax diagrams contain many different items. Syntax items include:

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

**Note:** If a syntax diagram shows a character that is not alphanumeric (for example, parentheses, periods, commas, equal signs, a blank space), enter the character as part of the syntax.
Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

<table>
<thead>
<tr>
<th>Item type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>Required items are displayed on the main path of the horizontal line.</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional items are displayed below the main path of the horizontal line.</td>
</tr>
<tr>
<td>Default</td>
<td>Default items are displayed above the main path of the horizontal line.</td>
</tr>
</tbody>
</table>

**Syntax examples**

The following table provides syntax examples.

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required item.</td>
<td></td>
</tr>
</tbody>
</table>

Required items appear on the main path of the horizontal line. You must specify these items.

Required choice.

A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.

Optional item.

Optional items appear below the main path of the horizontal line.

Optional choice.

An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.

Default.

Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.

Variable.

Variables appear in lowercase italics. They represent names or values.
Table 2. Syntax examples (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeatable item.</td>
<td>![repeatable_item]</td>
</tr>
<tr>
<td>An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.</td>
<td>![repeatable_item]</td>
</tr>
<tr>
<td>A character within the arrow means you must separate repeated items with that character.</td>
<td>![repeatable_item]</td>
</tr>
<tr>
<td>An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.</td>
<td>![repeatable_item]</td>
</tr>
<tr>
<td>Fragment.</td>
<td>![fragment]</td>
</tr>
<tr>
<td>The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.</td>
<td>![fragment]</td>
</tr>
</tbody>
</table>

Where to find more information

For an overview of the information associated with z/OS, see [z/OS Information Roadmap](http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR).

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and documentation APARs for z/OS, see the online document at: [http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR](http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR).

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS.

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required): [http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp](http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp)
How to send your comments to IBM

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

Use one of the following methods to send us your comments:

1. Send an email to mhvrdfs@us.ibm.com
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRDFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   U.S.A.
4. Fax the comments to us as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:
• Your name and address
• Your email address
• Your telephone or fax number
• The publication title and order number:
  z/OS V1R13.0 Language Environment Programming Reference
  SA22-7562-13
• The topic and page number related to your comment
• The text of your comment.

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

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

If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:
• Contact your IBM service representative
• Call IBM technical support
• Visit the IBM support portal at http://www.ibm.com/systems/z/support/
Summary of changes

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes made in z/OS Version 1 Release 13

The book contains information that was previously presented in z/OS Language Environment Programming Reference, SA22-7562-12, which supports z/OS Version 1 Release 12.

New information:

- The new run-time option, PAGEFRAMESIZE, allows for the HEAP, ANYHEAP, or upward-growing STACK page frame size to be selectively configured for AMODE 31 applications. See "PAGEFRAMESIZE" on page 58 for more information.
- Information about three countries has been added to the list of IBM-supplied country code defaults; see Table 32 on page 515 for details.
- Additional usage information has been added to callable service "CEEENV—Process environmental variables" on page 277.

Changed information:

- The sys/subsys parameter of CEE3INF no longer supports bit 31; see "CEE3INF—Query enclave information" on page 169 for details.
- CEEGPID had been updated to reflect the current release level of Language Environment; see "CEEGPID—Retrieve the Language Environment version and platform ID" on page 316 for details.

Changes made in z/OS Version 1 Release 12


New information:

- z/OS XL C/C++ supports MSGFILE and CEEDUMP DDNAMEs that have been dynamically allocated with the XTIOT, UCB nocapture, and DSAB-above-the-line options specified in the SVC99 parameters.
- Information about the environment variable _CEE_REALLOC_CONTROL has been added to the CECECZST callable service (see "CEECZST—Reallocate (change size of) storage" on page 232). This variable provides improvements to more efficiently use storage.

Changed information:

- Examples of CEE3AB2 have been updated; see "CEE3AB2—Terminate enclave with an abend and reason code" on page 124 for more information.
- The time stamp information included in the names of dynamic data sets that are produced by the DYNDUMP run-time option has been clarified; see "DYNDUMP" on page 26 for more information.
• The "Readers’ Comments - We’d Like to Hear from You" section at the back of this publication has been replaced with a new section “How to send your comments to IBM” on page xvii. The hardcopy mail-in form has been replaced with a page that provides information appropriate for submitting readers comments to IBM.

Deleted information:
• Information about changing the form-name if you no longer want it specified has been removed from the SYSOUT parameter in "CEEDUMP" on page 19.

Changes made in z/OS Version 1 Release 11


Changed information:
• Describes the possible values of CEE_Version_ID for CEEGPID.
• Allowing the user to set an specific number of entries of an specific pool ID in the HEAPCHK run-time option to format Heappool Trace Table.
Part 1. Language Environment Run-Time Options

The following sections contain detailed information about how to use the Language Environment run-time options.
Chapter 1. Language Environment run-time option reference tables and general information

This chapter includes quick reference tables for each of the Language Environment run-time options and information to help you read syntax diagrams and specify the run-time options. The tables list the location of the options and briefly state their function.

For detailed descriptions, refer to their respective chapters in this book.

Quick reference table for AMODE 31 run-time options

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

Table 3. Run-Time Options Quick Reference - AMODE 31

<table>
<thead>
<tr>
<th>Run-Time Options</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABPERC</td>
<td>Percolates a specified abend.</td>
<td>9</td>
</tr>
<tr>
<td>ABTERMENC</td>
<td>Sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater.</td>
<td>10</td>
</tr>
<tr>
<td>AIXBLD</td>
<td>NOAIXBLD</td>
<td>Invokes the access method services (AMS) for VSAM indexed and relative data sets to complete the file and index definition procedures for COBOL routines.</td>
</tr>
<tr>
<td>ALL31</td>
<td>Indicates whether an application does or does not run entirely in AMODE(31).</td>
<td>12</td>
</tr>
<tr>
<td>ANYHEAP</td>
<td>Controls allocation of library heap storage not restricted to below the 16M line.</td>
<td>14</td>
</tr>
<tr>
<td>ARGPARSE</td>
<td>Specifies if arguments on the command line are to be parsed in the usual C format.</td>
<td>15</td>
</tr>
<tr>
<td>AUTOTASK</td>
<td>Specifies if Fortran Multitasking Facility is to be used by your program and the number of tasks that are allowed to be active.</td>
<td>16</td>
</tr>
<tr>
<td>BELOWHEAP</td>
<td>Controls allocation of library heap storage below the 16M line.</td>
<td>17</td>
</tr>
<tr>
<td>CBLOPTS</td>
<td>Specifies the format of the argument string on application invocation when the main program is COBOL.</td>
<td>18</td>
</tr>
<tr>
<td>CBLPSHPPOP</td>
<td>Controls if CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL subprogram is called.</td>
<td>18</td>
</tr>
<tr>
<td>CBLQDA</td>
<td>Controls COBOL QSAM dynamic allocation.</td>
<td>19</td>
</tr>
<tr>
<td>CEEDUMP</td>
<td>Specifies options to control the processing of the Language Environment dump report.</td>
<td>19</td>
</tr>
<tr>
<td>CHECK</td>
<td>Indicates if “checking errors” within an application should be detected.</td>
<td>22</td>
</tr>
<tr>
<td>COUNTRY</td>
<td>Specifies the default formats for date, time, currency symbol, decimal separator, and the thousands separator based on a country.</td>
<td>22</td>
</tr>
<tr>
<td>DEBUG</td>
<td>NODEBUG</td>
<td>Activates the COBOL batch debugging features specified by the “debugging lines” or the USE FOR DEBUGGING declarative.</td>
</tr>
<tr>
<td>DEPTHCONDLMT</td>
<td>Limits the extent to which conditions can be nested.</td>
<td>24</td>
</tr>
<tr>
<td>Run-Time Options</td>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>DYNDUMP</td>
<td>Provides a way to obtain IPCS readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSEABEND DD statement</td>
<td>26</td>
</tr>
<tr>
<td>ENV</td>
<td>Specifies the operating system that your C application runs under.</td>
<td>28</td>
</tr>
<tr>
<td>ENVAR</td>
<td>Sets the initial values for the environment variables.</td>
<td>29</td>
</tr>
<tr>
<td>ERRCOUNT</td>
<td>Specifies how many conditions of severity 2, 3, and 4 can occur per thread before an enclave terminates abnormally.</td>
<td>31</td>
</tr>
<tr>
<td>ERRUNIT</td>
<td>Identifies the unit number to which run-time error information is to be directed.</td>
<td>32</td>
</tr>
<tr>
<td>EXECOPS</td>
<td>Specifies if run-time options can be specified on the command line.</td>
<td>32</td>
</tr>
<tr>
<td>FILEHIST</td>
<td>FILEHIST specifies whether to allow the file definition of a file referred to by a ddname to be changed during run time.</td>
<td>33</td>
</tr>
<tr>
<td>FILETAG</td>
<td>FILETAG ensures control of untagged HFS files and standard streams terminal files when set up for conversion, and for control of certain open functions which tag new or empty HFS files.</td>
<td>34</td>
</tr>
<tr>
<td>FLOW</td>
<td>NOFLOW</td>
<td>Controls the FLOW output produced by OS/VS COBOL programs.</td>
</tr>
<tr>
<td>HEAP</td>
<td>Controls allocation of the user heap.</td>
<td>36</td>
</tr>
<tr>
<td>HEAPCHK</td>
<td>Specifies that user heap storage can be inspected for damage and request a heap storage diagnostics report.</td>
<td>40</td>
</tr>
<tr>
<td>HEAPPOOLS</td>
<td>Specifies that (Quick) heap storage manager can be used.</td>
<td>42</td>
</tr>
<tr>
<td>INFOMSGFILTER</td>
<td>Allows the user to eliminate unwanted informational messages.</td>
<td>46</td>
</tr>
<tr>
<td>INQPCOPN</td>
<td>NOINQPCOPN</td>
<td>INQPCOPN controls if the OPENED specifier on an INQUIRE by unit statement can be used to determine whether a preconnected unit has had any I/O statements directed to it.</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>Causes attentions recognized by the host operating system to be recognized by Language Environment.</td>
<td>47</td>
</tr>
<tr>
<td>LIBSTACK</td>
<td>Controls the allocation of the thread’s library stack storage.</td>
<td>51</td>
</tr>
<tr>
<td>MSGFILE</td>
<td>Specifies the <em>ddname</em> of the run-time diagnostics file.</td>
<td>52</td>
</tr>
<tr>
<td>MSGQ</td>
<td>Specifies the number of ISI blocks allocated on a per-thread basis during execution.</td>
<td>55</td>
</tr>
<tr>
<td>NATLANG</td>
<td>Specifies the national language to use for the run-time environment.</td>
<td>55</td>
</tr>
<tr>
<td>OCSTATUS</td>
<td>NOOCSTATUS</td>
<td>Controls if the OPEN and CLOSE status specifiers are verified.</td>
</tr>
<tr>
<td>PAGEFRAMESIZE</td>
<td>Indicates the preferred page frame size for certain types of storage requests. Note the statement of direction associated with this run-time option.</td>
<td>58</td>
</tr>
<tr>
<td>PC</td>
<td>Specifies that Fortran static common blocks are not shared among load modules.</td>
<td>60</td>
</tr>
<tr>
<td>PLIST</td>
<td>Specifies the format of the invocation arguments received by your C application when it is invoked.</td>
<td>60</td>
</tr>
<tr>
<td>PLITASKCOUNT</td>
<td>Controls the maximum number of tasks active at one time while you are running PL/I MTF applications.</td>
<td>62</td>
</tr>
<tr>
<td>POSIX</td>
<td>Specifies if the enclave can run with the POSIX semantics.</td>
<td>62</td>
</tr>
<tr>
<td>PROFILE</td>
<td>Controls the use of an optional profiler which collects performance data for the running application.</td>
<td>63</td>
</tr>
</tbody>
</table>
Table 3. Run-Time Options Quick Reference - AMODE 31 (continued)

<table>
<thead>
<tr>
<th>Run-Time Options</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRTUNIT</td>
<td>Identifies the unit number used for PRINT and WRITE statements that do not specify a unit number.</td>
<td>64</td>
</tr>
<tr>
<td>PUNUNIT</td>
<td>Identifies the unit number used for PUNCH statements that do not specify a unit number.</td>
<td>64</td>
</tr>
<tr>
<td>RDRUNIT</td>
<td>Identifies the unit number used for READ statements that do not specify a unit number.</td>
<td>65</td>
</tr>
<tr>
<td>REDIR</td>
<td>Specifies if redirections for stdin, stderr, and stdout are allowed from the command line.</td>
<td>66</td>
</tr>
<tr>
<td>RECPAD</td>
<td>Specifies if a formatted input record is padded with blanks.</td>
<td>65</td>
</tr>
<tr>
<td>RPTOPTS</td>
<td>Specifies that a report of the run-time options in use by the application be generated.</td>
<td>66</td>
</tr>
<tr>
<td>RPTSTG</td>
<td>Specifies that a report of the storage used by the application be generated at the end of execution.</td>
<td>68</td>
</tr>
<tr>
<td>RTEREUS</td>
<td>NORTEREUS</td>
<td>Initializes the run-time environment to be reusable when the first COBOL program is invoked.</td>
</tr>
<tr>
<td>SIMVRD</td>
<td>NOSIMVRD</td>
<td>Specifies if your COBOL programs use a VSAM KSDS to simulate variable length relative organization data sets.</td>
</tr>
<tr>
<td>STACK</td>
<td>Controls the allocation and management of stack storage.</td>
<td>71</td>
</tr>
<tr>
<td>STORAGE</td>
<td>Controls the contents of storage that is allocated and freed.</td>
<td>75</td>
</tr>
<tr>
<td>TERMTHDACT</td>
<td>Sets the level of information produced due to an unhandled error of severity 2 or greater.</td>
<td>78</td>
</tr>
<tr>
<td>TEST</td>
<td>NOTEST</td>
<td>Specifies that a debug tool is to be given control according to the suboptions specified.</td>
</tr>
<tr>
<td>THREADHEAP</td>
<td>Controls the allocation and management of thread-level heap storage.</td>
<td>86</td>
</tr>
<tr>
<td>THREADSTACK</td>
<td>Controls the allocation of thread-level stack storage for both the upward and downward-growing stack.</td>
<td>88</td>
</tr>
<tr>
<td>TRACE</td>
<td>Determines if Language Environment run-time library tracing is active.</td>
<td>91</td>
</tr>
<tr>
<td>TRAP</td>
<td>Specifies how Language Environment routines handle abends and program interrupts.</td>
<td>93</td>
</tr>
<tr>
<td>UPSI</td>
<td>Sets the eight UPSI switches on or off.</td>
<td>95</td>
</tr>
<tr>
<td>USRHDLR</td>
<td>Registers user condition handlers.</td>
<td>96</td>
</tr>
<tr>
<td>VCTRSAVE</td>
<td>Specifies if any language in an application uses the vector facility when user-written condition handlers are called.</td>
<td>97</td>
</tr>
<tr>
<td>XPLINK</td>
<td>Controls the initialization of the XPLINK environment.</td>
<td>98</td>
</tr>
<tr>
<td>XUFLOW</td>
<td>Specifies if an exponent underflow causes a program interrupt.</td>
<td>100</td>
</tr>
</tbody>
</table>

Quick reference table for AMODE 64 run-time options

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

Table 4. Run-Time Options Quick Reference - AMODE 64

<table>
<thead>
<tr>
<th>Run-Time Options</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGPARSE</td>
<td>Specifies if arguments on the command line are to be parsed in the usual C format.</td>
<td>15</td>
</tr>
<tr>
<td>Run-Time Options</td>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>CEEDUMP</td>
<td>Specifies options to control the processing of the Language Environment dump report.</td>
<td>19</td>
</tr>
<tr>
<td>DYNDUMP</td>
<td>Provides a way to obtain IPCS readable dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement.</td>
<td>26</td>
</tr>
<tr>
<td>ENVAR</td>
<td>Sets the initial values for the environment variables.</td>
<td>29</td>
</tr>
<tr>
<td>EXECOPS</td>
<td>Specifies whether run-time options can be specified on the command line.</td>
<td>32</td>
</tr>
<tr>
<td>FILETAG</td>
<td>FILETAG ensures control of untagged HFS files and standard streams terminal files when set up for conversion, and for control of certain open functions which tag new or empty HFS files.</td>
<td>34</td>
</tr>
<tr>
<td>HEAP64</td>
<td>Controls allocation of the user heap.</td>
<td>38</td>
</tr>
<tr>
<td>HEAPCHK</td>
<td>Specifies that user heap storage can be inspected for damage and request a heap storage diagnostics report.</td>
<td>40</td>
</tr>
<tr>
<td>HEAPPPOOLS</td>
<td>Specifies that the (Quick) heap storage manager can be used to manage user heap storage above the 16M line and below the 2G bar.</td>
<td>42</td>
</tr>
<tr>
<td>HEAPPPOOLS64</td>
<td>Specifies that the (Quick) heap storage manager can be used to manage user heap storage above the 2G bar.</td>
<td>44</td>
</tr>
<tr>
<td>INFOMSGFILTER</td>
<td>Allows the user to eliminate unwanted informational messages.</td>
<td>46</td>
</tr>
<tr>
<td>IOHEAP64</td>
<td>Controls allocation of I/O heap storage.</td>
<td>48</td>
</tr>
<tr>
<td>LIBHEAP64</td>
<td>Controls allocation of library heap storage.</td>
<td>49</td>
</tr>
<tr>
<td>NATLANG</td>
<td>Specifies the national language to use for the run-time environment.</td>
<td>55</td>
</tr>
<tr>
<td>POSIX</td>
<td>Specifies if the enclave can run with the POSIX semantics.</td>
<td>62</td>
</tr>
<tr>
<td>PROFILE</td>
<td>Controls the use of an optional profiler which collects performance data for the running application.</td>
<td>63</td>
</tr>
<tr>
<td>REDIR</td>
<td>Specifies if redirections for stdin, stderr, and stdout are allowed from the command line.</td>
<td>66</td>
</tr>
<tr>
<td>RTPOPTS</td>
<td>Specifies that a report of the run-time options in use by the application be generated.</td>
<td>66</td>
</tr>
<tr>
<td>RPTSTG</td>
<td>Specifies that a report of the storage used by the application be generated at the end of execution.</td>
<td>68</td>
</tr>
<tr>
<td>STACK64</td>
<td>Controls the allocation and management of stack storage.</td>
<td>74</td>
</tr>
<tr>
<td>STORAGE</td>
<td>Controls the contents of storage that is allocated and freed.</td>
<td>75</td>
</tr>
<tr>
<td>TERMTHDACT</td>
<td>Sets the level of information produced due to an unhandled error of severity 2 or greater.</td>
<td>78</td>
</tr>
<tr>
<td>TEST / NOTEST</td>
<td>Specifies that a debug tool is to be given control according to the suboptions specified.</td>
<td>84</td>
</tr>
<tr>
<td>THREADSTACK64</td>
<td>Controls the allocation of thread-level stack storage for both the upward and downward-growing stack.</td>
<td>90</td>
</tr>
<tr>
<td>TRACE</td>
<td>Determines if Language Environment run-time library tracing is active.</td>
<td>91</td>
</tr>
<tr>
<td>TRAP</td>
<td>Specifies how Language Environment routines handle abends and program interrupts.</td>
<td>93</td>
</tr>
</tbody>
</table>
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 include the comma to indicate the omission of the suboption. For example: STACK(,,ANYWHERE,FREE,,). Trailing commas, however, are not required. STACK(,,ANYWHERE,FREE) is valid. If you do not specify any suboptions, they are ignored. Either of the following is valid: STACK or STACK().

If you run applications that are invoked by one of the exec family of functions, such as a program executed under the z/OS UNIX System Services shell, you can also use the _CEE_RUNOPTS environment variable to specify run-time options. Refer to z/OS Language Environment Programming Guide for a detailed description of the various ways in which you can specify Language Environment run-time options and program arguments, and use _CEE_RUNOPTS.

Restriction: 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.

Propagating run-time options with spawn and exec

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

Using the CEEOPTS DD statement

Language Environment allows you to provide additional invocation-level run-time options using the CEEOPTS DD statement. The CEEOPTS DD can refer to an in-stream data set, regular sequential data set, or a member of a regular or extended partitioned data set. If specified, the data set must be available during initialization of the enclave so the options can be merged. To specify the CEEOPTS DD statement, use the following syntax, as appropriate.

<table>
<thead>
<tr>
<th>For in-stream JCL</th>
<th>//CEEOPTS DD * ALL31(OFF),STACK(,,BELOW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>For a sequential data set</td>
<td>//CEEOPTS DD DSN=MY.CEEOPTS.DATASET,DISP=SHR</td>
</tr>
<tr>
<td>For a partitioned data set</td>
<td>//CEEOPTS DD DSN=MY.CEEOPTS.DATASET(MYOPTS), DISP=SHR</td>
</tr>
<tr>
<td>To ignore the DD statement</td>
<td>//CEEOPTS DD DUMMY</td>
</tr>
</tbody>
</table>

Before using the CEEOPTS DD statement, review the following restrictions:
1. The CEEOPTS DD supports only DASD data sets that can be read with QSAM. An informational message is issued when an unsupported data set type is used.
2. Only the first 3K (excluding comment lines) of the CEEOPTS file are read. All other information is ignored.
3. The file must be in fixed-block or fixed format. Variable block format is not supported.
4. The CEEOPTS DD is ignored under CICS, SPC, and for an exec()ed program.
5. Language Environment cannot use a CEEOPTS DDNAME dynamically allocated with the XTIOT, UCB nocapture, or DSAB-above-the-line options specified in the SVC99 parameters (S99TIOEX, S99ACUCB, S99DSABA flags).

As Figure 1 shows, the syntax of the run-time options within the CEEOPTS DD statement is similar to the syntax used on the JCL PARM= parameter or under TSO.
- Each record in the CEEOPTS DD is concatenated to form one options string. No implicit space is added between lines.
- Options must be separated by commas or blanks and can span multiple lines.
- A trailing slash (/) is not valid to denote the end of the options string.
- Input lines that begin with an asterisk (*) in column 1 are treated as comments and ignored.
- The last 8 columns of each record are treated as sequential information and ignored.

```
****** **** TOP OF FILE ***
000001 * This line is a comment
000002 ALL31(OFF),STACK(,,BELOW)
000003 TRAP(ON,
000004 NOSPIE
000005 * This line is a comment within an option
000006 ),TERMTHDACT(
000007 UAIMM,
000008 CICSDDS,96)
****** *** END OF DATA ***
```

Figure 1. Example syntax for the CEEOPTS DD statement

The Language Environment run-time options report identifies the options merged from the CEEOPTS DD source by using DD:CEEOPTS as an indicator. For an example of the options report, see z/OS Language Environment Debugging Guide.
Chapter 2. Using the Language Environment run-time options

This chapter describes the Language Environment run-time options, their syntax, and their usage.

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

**ABPERC**

**Derivation**

**ABnormal PERColation**

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

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

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

<table>
<thead>
<tr>
<th>Default Value</th>
<th>CICS</th>
<th>ABPERC is ignored under CICS.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-CICS</strong></td>
<td>ABPERC(NONE)</td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```
...ABPERC (NONE abcode)...
```
ABPERC

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

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

- **Shhh** A system abend code, where `hhh` is the hex system abend code
- **Udddd** A user abend code, where `dddd` is a decimal user abend code. Any 4-character string can also be used as `dddd`.

**z/OS UNIX consideration**

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

**Usage notes**

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

ABTERMENC

**Derivation**

ABnormal TERmination of the ENClave

ABTERMENC sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater.

**Restriction:** TRAP(ON) must be in effect for ABTERMENC to have an effect.

**Default Value**

- **Non-CICS** ABTERMENC(ABEND)
- **CICS** ABTERMENC(ABEND)

**Syntax**

```
ABTermenc (ABEND RETCODE)
```

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

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

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

z/OS UNIX consideration

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

Usage notes

- You can use the assembler user exit, CEEAUE_ABND, to modify the behavior of this run-time option by setting the CEEAUE_ABND flag. See z/OS Language Environment Programming Guide for more information.
- To gather information about the unhandled condition, see “TERMTHDACT” on page 78.

For more information

For information about return code calculation and abend codes, see z/OS Language Environment Programming Guide.

AIXBLD (COBOL Only)

Derivation

Alternate IndeX BuILD

AIXBLD invokes the access method services (AMS) for VSAM indexed and relative data sets (KSDS and RRDS) to complete the file and index definition procedures for COBOL programs. AIXBLD conforms to the ANSI 1985 COBOL standard.

Default Value

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>NOAIXBLD</td>
</tr>
<tr>
<td>CICS</td>
<td>AIXBLD is ignored under CICS.</td>
</tr>
</tbody>
</table>

Syntax

NOAIXBLD
  NOAIX
  AIXBLD
  AIX

NOAIXBLD | NOAIX

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

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

**z/OS UNIX consideration**
If you also specify the MSGFILE run-time option, the access method services messages are directed to the MSGFILE *ddname* or to the default SYSOUT.

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

**For more information**
- See "MSGFILE" on page 52 for information about the MSGFILE run-time option.

---

**ALL31**

<table>
<thead>
<tr>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL AMODE 31</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS: ALL31(ON)</td>
</tr>
<tr>
<td>CICS: ALL31(ON)</td>
</tr>
</tbody>
</table>

**Syntax**

```
ALL31 (ON, OFF)
```

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

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

When ALL31(ON) remains in effect:

- AMODE switching across calls to Language Environment common run-time routines is minimized. For example, no AMODE switching is performed on calls to Language Environment callable services.
- Language Environment allocates storage for the common anchor area (CAA) and other control blocks in unrestricted storage.
- COBOL EXTERNAL data is allocated in unrestricted storage.

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

- Language Environment uses more storage below the 16M line.
- AMODE switching across calls to Language Environment common run-time routines is performed. For example, AMODE switching is performed on calls to Language Environment callable services.
- Language Environment allocates storage for the common anchor area (CAA) and other control blocks in storage below the 16M line.
- COBOL EXTERNAL data is allocated in storage below the 16M line.

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

z/OS UNIX considerations

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

Usage notes

- Restrictions: You must specify ALL31(OFF) if your COBOL applications contain one of the following programs:
  - VS COBOL II NORES
  - OS/VS COBOL (non-CICS)
  - If the Language Environment environment was initialized using ILBSTP0
- PL/I considerations: For PL/I MTF applications, Language Environment provides AMODE switching. Therefore, the first routine of a task can be in AMODE 24.
- Fortran considerations: Use ALL31(ON) if all of the compile units in the enclave have been compiled with VS FORTRAN Version 1 or Version 2 and there are no requirements for 24-bit addressing mode. Otherwise, use ALL31(OFF).
- XPLINK considerations: When an application is running in an XPLINK environment (that is, either the XPLINK(ON) run-time option was specified, or the initial program contained at least one XPLINK compiled part), the ALL31 run-time option is forced to ON. No AMODE 24 routines are allowed in an enclave that uses XPLINK. No message is issued to indicate this action. If a Language Environment run-time options report is generated using the RPTOPTS run-time option, the ALL31 option is reported as "Override" under the LAST WHERE SET column.
ALL31

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

**Performance Consideration**

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

**For More Information**

- See "STACK" on page 71 for information about the STACK run-time option.
- See "XPLINK" on page 98 for information about the XPLINK run-time option.

---

**ANYHEAP**

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

<table>
<thead>
<tr>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
</tr>
<tr>
<td>CICS</td>
</tr>
</tbody>
</table>

**Syntax**

```
ANYheap(\[init_size\],\[incr_size\],\[ANYWHERE\],\[FREE\])
```

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

- **incr_size**
  - Determines the minimum size of any subsequent increment to the anywhere heap. This value can be specified as \(n\), \(n\)K, or \(n\)M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

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

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

**CICS considerations**
- If ANYHEAP(0) is specified, the initial anywhere heap segment is obtained on the first use and will be based on the increment size.
- The maximum initial and increment size for the anywhere heap is 1 gigabyte (1024M).
- The default increment size is 4080 bytes, rather than 4096 bytes, to accommodate the 16 byte CICS storage check zone. Without this accommodation, an extra page of storage is allocated (only when the storage allocation is below the 16 MB line). If you choose to change the increment size, it is recommended that you adjust for the 16 byte CICS storage check zone.

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

**Performance consideration**
You can improve performance with the ANYHEAP run-time option by specifying values that minimize the number of times the operating system allocates storage. See [“RPTSTG” on page 68](#) for information on how to generate a report you can use to determine the optimum values for the ANYHEAP run-time option.

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

---

**ARGPARSE | NOARGPARSE (C only)**

<table>
<thead>
<tr>
<th>Derivation</th>
<th>ARGument PARSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGPARSE</td>
<td></td>
</tr>
<tr>
<td>NOARGPARSE</td>
<td></td>
</tr>
</tbody>
</table>

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

**Restriction:** You cannot set ARGPARSE during Language Environment installation.

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>ARGPARSE</td>
</tr>
<tr>
<td>CICS</td>
<td>ARGPARSE is ignored under CICS.</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>ARGPARSE</td>
</tr>
</tbody>
</table>
ARGPARSE | NOARGPARSE

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

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

**Usage note**

You must specify ARGPARSE for the REDIR run-time option to have an effect.

**For more information**

See "REDIR | NOREDIR (C Only)" on page 66 for a description of REDIR.

AUTOTASK | NOAUTOTASK (Fortran only)

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

<table>
<thead>
<tr>
<th>Default Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS NOAUTOTASK</td>
<td></td>
</tr>
<tr>
<td>CICS AUTOTASK is ignored under CICS.</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```
AUTOTASK

NOAUTOTASK

(loadmod, numtasks)
```

NOAUTOTASK

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

**loadmod**

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

**numtasks**

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

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

**Default Value**

- **Non-CICS**: BELOWHEAP(8K,4K,FREE)
- **CICS**: BELOWHEAP(4K,4080,FREE)

**Syntax**

```
BELOWHEAP (init_size, incr_size, FREE|KEEP)
```

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

- **incr_size**: Determines the minimum size of any subsequent increment to the area below the 16M line, and is specified in \( n \), \( n \)K, or \( n \)M bytes of storage. This value is rounded up to the nearest multiple of 8 bytes.

- **FREE**: Specifies that storage allocated to BELOWHEAP increments is released when the last of the storage is freed.

- **KEEP**: Specifies that storage allocated to BELOWHEAP increments is not released when the last of the storage within that increment is freed.

**CICS consideration**

The default increment size is 4080 bytes, rather than 4096 bytes, to accommodate the 16 byte CICS storage check zone. Without this accommodation, an extra page of storage is allocated (only when the storage allocation is below the 16 MB line). If you choose to change the increment size, it is recommended that you adjust for the 16 byte CICS storage check zone.

**z/OS UNIX considerations**

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

**Usage note**

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

**Performance consideration**

You can improve performance with the BELOWHEAP run-time option by specifying values that minimize the number of times the operating system allocates...
BELOWHEAP

storage. See "RPTSTG" on page 68 for information on how to generate a report you can use to determine the optimum values for the BELOWHEAP run-time option.

For more information

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

CBLOPTS (COBOL only)

Derivation

CBLOPTS

COBOL Options

CBLOPTS specifies the format of the parameter string on application invocation when the main program is COBOL. CBLOPTS determines if run-time options or program arguments appear first in the parameter string. Because you can specify this option only in CEEUOPT, CEEROPT or CEEDOPT at initialization, see z/OS Language Environment Customization for details.

CBLPSHPOP (COBOL only)

Derivation

CBLPSHPOP

COBOL Push Pop

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

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

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

Default Value

Non-CICS CBLPSHPOP is ignored.
CICS CBLPSHPOP(ON)

Syntax

CBLPSHPOP(ON)

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.
CBLPSHPOP

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

**Performance consideration**

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

**For more information**

- For more information about CEEUOPT, see [z/OS Language Environment Customization](#).
- To determine where to locate information about CICS commands, see *CICS Master Index*.

**CBLQDA (COBOL only)**

<table>
<thead>
<tr>
<th>Derivation</th>
<th>COBOL QSAM Dynamic Allocation</th>
</tr>
</thead>
</table>

CBLQDA controls COBOL QSAM dynamic allocation on an OPEN statement.

**Default**

<table>
<thead>
<tr>
<th>Non-CICS</th>
<th>CBLQDA(OFF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>CBLQDA is ignored under CICS.</td>
</tr>
</tbody>
</table>

**Syntax**

```plaintext
CBLQda (OFF ON)
```

**OFF**  Specifies that COBOL QSAM dynamic allocation is not permitted.

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

**Usage notes**

- CBLQDA(OFF) is the recommended default because it prevents a temporary data set from being created in case there is a misspelling in your JCL. If you specify CBLQDA(ON) and have a misspelling in your JCL, Language Environment creates a temporary file, writes to it, and then z/OS deletes it. You receive a return code of 0, but no output.
- CBLQDA does not affect dynamic allocation for the message file specified in MSGFILE or the Language Environment formatted dump file (CEEDUMP).

**CEEDUMP**

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Common Execution Environment DUMP</th>
</tr>
</thead>
</table>

Chapter 2. Using the Language Environment run-time options  19
CEEDUMP

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC)</td>
</tr>
<tr>
<td>CICS</td>
<td>CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC)</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC)</td>
</tr>
</tbody>
</table>

**Syntax**

```
CEEDump(
  page_len,
  SYSOUT=
    class,
    (class),
    FREE=END,
    FREE=CLOSE,
    SPIN=UNALLOC
)
```

*page_len*

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

*SYSOUT=*  
Specifies SYSOUT attributes for a dynamically allocated CEEDUMP DD. SYSOUT has three possible parameters, but the second parameter should always be omitted:

  * **class**
    Specifies a value that is one character in length. Valid values are A through Z, 0 through 9, and *. A SYSOUT *class* must not be specified inside quotation marks.
      - If *class* is not specified, it defaults to * for the dynamically allocated CEEDUMP.
      - If dynamic allocation for the specified SYSOUT class specified fails, SYSOUT=* is set and message CEE378S is issued.

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

*FREE=*  
Specifies that dynamically allocated CEEDUMP's will have one of the following JCL DD attributes:
CEEDUMP

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

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

**SPIN=**  Specifies that dynamically allocated CEEDUMPs will have one of the following attributes:

- **UNALLOC**  The SPIN=UNALLOC attribute indicates the system should make the SYSOUT data set available for processing immediately when it is unallocated. This is the default value for this suboption.

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

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

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

**Usage notes**

- If a CEEDUMP DD card is explicitly coded in a job step, Language Environment ignores any SYSOUT class, FREE, SPIN or form-name specified in the CEEDUMP run-time.

- The SYSOUT=class suboption is overridden by _CEE_DMPTARG when this environment variable is used at the same time to indicate the SYSOUT class.

- The page_len suboption is overridden by the CEE3DMP PAGESIZE option. For more information about CEE3DMP, see [z/OS Language Environment Customization](z/OS Language Environment Customization).

- Language Environment supports the use of a CEEDUMP DDNAME dynamically allocated with the XTIOT, UCB nocapture, or DSAB-above-the-line options specified in the SVC99 parameters (S99TIOEX, S99ACUCB, S99DSABA flags).

**Examples**

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

- **CEEDUMP(40)**  The CEEDUMP report contains a page length of 40 lines. The other implicit suboptions are SYSOUT=, FREE=, and SPIN=, which default to whatever values are currently set in the CEEDUMP run-time option.
CHECK

CHECK (COBOL only)

If your COBOL application has been compiled with the SSRANGE compile option, specifying the CHECK(ON) run-time option enables range checking of each index, subscript, and reference modification.

Default Value

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>CHECK(ON)</td>
</tr>
<tr>
<td>CICS</td>
<td>CHECK(ON)</td>
</tr>
</tbody>
</table>

Syntax

```plaintext
CHECK (ON | OFF)
```

- **ON** Specifies that run-time range checking is performed.
- **OFF** Specifies that run-time range checking is not performed.

Usage note

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

Performance consideration

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

COUNTRY

Specifying a `country_code` with the COUNTRY run-time option determines:

- date and time formats
- the currency symbol
- the decimal separator
- the thousands separator

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

Tip: You can set the country value using the COUNTRY run-time option or the CEE3CCTY callable service.

Default Value

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>COUNTRY(US)</td>
</tr>
<tr>
<td>CICS</td>
<td>COUNTRY(US)</td>
</tr>
</tbody>
</table>

Syntax

```plaintext
COUNTRY (country_code)
```
**COUNTRY**

`country_code`
A 2-character code that indicates to Language Environment the country on which to base the default settings. Table 32 on page 515 contains a list of valid country codes.

**Usage notes**

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

If an unsupported `country_code` is specified on the COUNTRY run-time option and one of the services listed in “National Language Support callable services” on page 109 is called with the optional `country_code` parameter omitted, the called service returns its specified default value.

- Language Environment provides locales used in C and C++ to establish default formats for the locale-sensitive functions and locale callable services, such as date and time formatting, sorting, and currency symbols. The COUNTRY setting does not affect these locale-sensitive functions and locale callable services.

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

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

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

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

**For more information**

- For more information about the CEE3CTY callable service, see "CEE3CTY—Set default country" on page 132.
- For more information about the RPTOPTS run-time option, see "RPTOPTS" on page 66.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 68.
- For a list of countries and their codes, see Appendix A, “IBM-supplied country code defaults,” on page 515.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 448.
- For more information on `setlocale()`, see z/OS XL C/C++ Programming Guide.

**DEBUG (COBOL only)**

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>NODEBUG</td>
</tr>
<tr>
<td>CICS</td>
<td>NODEBUG</td>
</tr>
</tbody>
</table>
**DEBUG**

**Syntax**

```
NODEBUG
DEBUG
```

**NODEBUG**

Suppresses the COBOL batch debugging features.

**DEBUG**

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

**Usage notes**

- Use DEBUG and NODEBUG only on the command line.
- For information on specifying this option in CEEDOPT (CEECOPT), CEEROPT or CEEUOPT, see [z/OS Language Environment Customization](#).

**Performance consideration**

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

**For more information**


**DEPTCHCONDLMT**

**Derivation**

```
DEPTH of nested CONDition LiMiT
```

DEPTCHCONDLMT specifies the extent to which conditions can be nested. Figure 2 on page 25 illustrates the effect of DEPTCHCONDLMT(3) on condition handling. The initial condition and two nested conditions are handled in this example. The third nested condition is not handled.
**DEPTHCONDLMT**

```
Error
  (level 1)
  User-written condition handler
  Another error
  (level 2)
  User-written condition handler
  Another error
  (level 3)
  User-written condition handler
  Another error
  (level 4)
  Not handled
```

*Figure 2. Effect of DEPTHCONDLMT(3) on Condition Handling*

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>DEPTHCONDLMT(10)</td>
</tr>
<tr>
<td>CICS</td>
<td>DEPTHCONDLMT(10)</td>
</tr>
</tbody>
</table>

**Syntax**

```
DEPTHCONDLMT(limit)
```

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

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

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

**z/OS UNIX consideration**

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

**Usage notes**

- PL/I consideration: DEPTHCONDLMT(0) provides PL/I compatibility.
- PL/I MTF consideration: In a PL/I MTF application, DEPTHCONDLMT sets the limit for how many nested synchronous conditions are allowed for a PL/I task. If the number of nested conditions exceeds the limit, the application ends abnormally.

**For more information**

For more information on nested conditions, see [z/OS Language Environment Programming Guide](#).
DYNDUMP

Derivation

DYNamic DUMP

The DYNDUMP run-time option provides a way to obtain dynamic dumps of user applications that would ordinarily be lost due to the absence of a SYSMDUMP, SYSUDUMP, or SYSABEND DD statement. The dynamic dump is written when:

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

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

Default Value

Non-CICS
DYNDUMP(*userid,NODYNAMIC,TDUMP)

CICS
DYNDUMP is ignored.

AMODE 64
DYNDUMP(*userid,NODYNAMIC,TDUMP)

Syntax

```markdown
DYNDump({*USERID
| hlq
| *TSOPREFIX}), NODYNAMIC, DYNAMIC, FORCE, TDUMP, NOTDUMP}
```

hlq is a high level qualifier for the dynamic dump data set to be created. This is concatenated with a time stamp consisting of the Julian day and the time of the dump. The job name or PID can also be part of the name if the combined length of hlq and the time stamp is 35 characters or less. The hlq value is limited to 26 characters including dot (.) separators. hlq allows three keywords:

*USERID

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

*TSOPREFIX or TSOPRE

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

Each keyword may be followed by additional characters to be used to create the dataset name. When appended to the USERID or the TSO prefix, they form the hlq used when creating the dump data set.
The data set name is limited to 44 characters. If the combined length of hlq and the time stamp is 35 characters or less, the job name or PID is added to the data set name. If the system is using multilevel security, the SECLABEL is used as the second qualifier. If hlq contains multiple qualifiers, only the first is used, followed by the SECLABEL. The format of the data set name is:

- When the application is not exec()ed and not multilevel security:
  \texttt{hlq.Djjj.Thhmmsst.jobname}

- When the application is exec()ed and not multilevel security:
  \texttt{hlq.Djjj.Thhmmsst.Pppppppp}

- When the application is multilevel security and not exec()ed:
  \texttt{hlq.MLS-level.Djjj.Thhmmsst.jobname}

- When the application is both multilevel security and exec()ed:
  \texttt{hlq.MLS-level.Djjj.Thhmmsst.Pppppppp}

For U4039 ABENDS

The following suboptions are used for U4039 ABENDS only:

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

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

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

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

For U40xx ABENDS

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

- **TDUMP**
  Language Environment creates a dynamic dump automatically.

- **NOTDUMP**
  Language Environment does not create a dynamic dump.

**Usage Notes**

- **Recommendations:**
  - Set up an hlq to which everyone can write.
  - Do not use FORCE or BOTH as the default for the U4039 ABENDS.

- The DYNDUMP run-time option is ignored under CICS.

- When an ABEND occurs during Language Environment initialization, the dynamic dump is not created if run-time options have not been processed yet.

- When the dynamic dump fails, messages are written to the operator's console or the job log (for batch). These are written by the IEATDUMP system service, by Language Environment, or by RACF®.
DYNDUMP

- When an ABEND has been issued without the DUMP option, no dump is generated.
- When Language Environment terminates with a U4038 abend, the U4038 abend is issued without the DUMP option. Therefore, no system dump is generated, and DYNDUMP does not collect a dump for this ABEND.
- The job name is taken from the &JOBNAME system symbol.
  - A dump for a TSO application uses the user ID of the &JOBNAME.
  - For a batch job, &JOBNAME is taken from the JOB card in the JCL.
  - In the shell, &JOBNAME is the user ID with a suffix.

Examples

- DYNDUMP(smith, force, notdump)
  A dynamic dump is generated only for ABEND code U4039. Other SYSnnnnn DD cards are ignored. Other ABENDs might cause a dump to be created if a SYSnnnnn DD card exists. The dynamic dump data set name will be similar to SMITH.D012.T112245.JOB11.
- DYNDUMP(smith,DYNAMIC,TDUMP)
  A dynamic dump is created if no SYSnnnnn is specified and the ABEND code is U4039. The data set name will be similar to SMITH.D117.T235900.JOBZ2.
- DYNDUMP(*TSOPREFIX,NODYNAMIC,TDUMP)
  A dynamic dump is generated only for ABEND code U40xx. The data set name will be similar to SMITH.D287.T234560.JOBZ2.
- DYNDUMP(*USERID,NODYNAMIC,TDUMP)
  A dynamic dump for a U4039 ABEND is taken to SMITH.D109.T234512.JOBZ3.
- DYNDUMP(*USERID.HOT.DUMPS,NODYNAMIC,TDUMP)
- DYNDUMP(*TSOPRE.A1234567.B1234567,NODYNAMIC,TDUMP)

ENV (C only)

<table>
<thead>
<tr>
<th>Derivation</th>
<th>ENVironment</th>
</tr>
</thead>
</table>

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

Restriction: This option does not apply to non-C languages and can be specified only with the C #pragma runopts directive. You cannot set ENV during Language Environment installation.

Default | Value
---|---
Non-CICS | The ENV option differs from other run-time options in that it does not have a standard default. The default depends on the system (CMS, IMS, or MVS) in which compilation occurs.
CICS | ENV is ignored under CICS.
**ENV**

### Syntax

```
ENV (CMS) (IMS) (MVS)
```

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

**z/OS UNIX consideration**

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

### ENVAR

**Derivation**

**ENvironmental VARiables**

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

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

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

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>ENVAR(&quot;&quot;)</td>
</tr>
<tr>
<td>CICS</td>
<td>ENVAR(&quot;&quot;)</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>ENVAR(&quot;&quot;)</td>
</tr>
</tbody>
</table>
ENVAR

Syntax

```
ENVAR(, string)
```

**string**

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

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

**z/OS UNIX consideration**

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

**Usage notes**

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

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

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

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

**For more information**

- For more information about the RPTOPTS run-time option, see “RPTOPTS” on page 66.
- For more information on `getenv()`, `putenv()`, `setenv()`, `clearenv()`, `system()`, and the exec and spawn family of functions, see z/OS XL C/C++ Run-Time Library Reference.
- For more information on the order of precedence of run-time option sources, see z/OS Language Environment Customization.
ERRCOUNT

**Derivation**

**ERRor COUNTer**

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

**Default Value**

- **Non-CICS**: ERRCOUNT(0)
- **CICS**: ERRCOUNT(0)

**Syntax**

```plaintext
>>>ERRcount(\[number\])
```

*number*

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

**z/OS UNIX consideration**

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

**Usage notes**

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

For more information
For more information in condition handling, see the z/OS Language Environment Programming Guide.

ERRUNIT (Fortran only)

<table>
<thead>
<tr>
<th>Derivation</th>
<th>ERRor UNIT</th>
</tr>
</thead>
</table>

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

Default Value
Non-CICS ERRUNIT(6)
CICS ERRUNIT is ignored under CICS.

Syntax

```
ERRUnit (number)
```

number
A number in the range 0-99.

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

EXECOPS | NOEXECOPS (C only)

EXECOPS specifies if you can enter run-time options on the command line.

Restriction: This option does not apply to non-C languages and can be specified only with the C #pragma runopts directive. You cannot set EXECOPS during Language Environment installation.

Default Value
Non-CICS EXECOPS
CICS This option is ignored under CICS.
AMODE 64 EXECOPS

Syntax

```
EXECOPS | NOEXECOPS
```

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

Language Environment parses the argument string to separate run-time options from application arguments. The run-time options are interpreted by Language Environment and the application arguments are passed to the application.

**NOEXECOPS**

Specifies that you cannot enter run-time options on the command line. Language Environment passes the entire argument string to the application.

**For more information**

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

---

**FILEHIST (Fortran only)**

**Derivation**

FILE HISTory

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

**Default Value**

Non-CICS: FILEHIST  
CICS: FILEHIST is ignored under CICS

**Syntax**

FILEHIST  
NOFILEHIST

**FILEHIST**

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

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

**NOFILEHIST**

Causes the history of a file to be disregarded in determining its existence. If you specify NOFILEHIST, you should consider:

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

FILETAG controls automatic text conversion and automatic file tagging for HFS files. It activates the automatic file tagging, on the first write, of new or empty z/OS UNIX file system files open with fopen() or freopen(), or upon the first I/O to a pipe created with popen(). You should be familiar with the concept of file tagging, autoconversion, and the program and file CCSIDs to use the run-time option properly. See [z/OS XL C/C++ Programming Guide](#) for more information.

### Default Value

**Non-CICS**
FILETAG(NOAUTOCVT,NOAUTOTAG)

**CICS**
FILETAG is ignored under CICS.

**AMODE 64**
FILETAG(NOAUTOCVT,NOAUTOTAG)

### Syntax

```plaintext
FILETAG (NOAUTOCVT, AUTOCVT), (NOAUTOTAG, AUTOTAG)
```

### NOAUTOCVT

Disables automatic text conversion as described below.

### AUTOCVT

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

**Restriction:** Automatic conversion can only take place between IBM-1047 and ISO8859-1 code sets. Other CCSID pairs are not supported for automatic text conversion. By default, automatic conversion for untagged UNIX file system files applies only to files opened in text mode. An untagged file opened in binary mode is not converted automatically. You can control this by using the _EDC_AUTOCVT_BINARY environment variable.

- If it is unset or set to NO, automatic conversion applies only to files opened in text mode.
- If it is set to YES, automatic conversion applies to files opened in either binary mode or text mode.

For more information on the _EDC_AUTOCVT_BINARY environment variable, see [z/OS XL C/C++ Programming Guide](#).

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

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

Requirement: The automatic text conversion is performed only if one of the following is also true:
• The _BPXK_AUTOCVT environment variable value is set to ON.
• The _BPXK_AUTOCVT environment variable is unset and AUTOCVT(ON) has been specified in the active BPXPRMxx member on your system.

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

NOAUTOTAG
Disables the automatic tagging of new or empty HFS files.

AUTOTAG
Enables automatic file tagging, on the first write, of new or empty HFS files opened with fopen(), freopen(), or popen().

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

FLOW controls the FLOW output produced by OS/VS COBOL programs. You cannot set FLOW during Language Environment installation.

Default Value

Non-CICS NOFLOW  
CICS NOFLOW

Syntax

NOFLOW
FLOW
FLOW(n)
FLOW=n
FLOWn

NOFLOW Suppresses the OS/VS COBOL FLOW output.
FLOW Allows the OS/VS COBOL FLOW output.

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

HEAP

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

- C data allocated as a result of the malloc(), calloc(), and realloc() functions
- COBOL WORKING-STORAGE data items
- PL/I variables with the storage class CONTROLLED, or the storage class BASED
- Data allocated as a result of a call to the CEEGTST Language Environment® callable service

Default Value

Non-CICS HEAP(32K,32K,ANYWHERE,KEEP,8K,4K)  
CICS HEAP(4K,4080,ANYWHERE,KEEP,4K,4080)

Syntax

Heap(Init_size,Incr_size,ANYWHERE,KEEP,FREE)
**init_size**
Determines the initial allocation of heap storage. This value can be specified as \( n \), \( n \times 10^3 \), or \( n \times 10^6 \) bytes of storage. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

**incr_size**
Determines the minimum size of any subsequent increment to the user heap storage. This value can be specified as \( n \), \( n \times 10^3 \), or \( n \times 10^6 \) bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

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

**BETWEEN**
Specifies that user heap storage is allocated below the 16M line in storage.

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

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

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

**initsz24**
Determines the minimum initial size of user heap storage that is obtained below the 16M line for AMODE 24 applications that specify ANYWHERE in the HEAP run-time option. This value can be specified as \( n \), \( n \times 10^3 \), or \( n \times 10^6 \) number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

**incrsz24**
Determines the minimum size of any subsequent increment to user heap storage that is obtained below the 16M line for AMODE 24 applications that specify ANYWHERE in the HEAP run-time option. This value can be specified as \( n \), \( n \times 10^3 \), or \( n \times 10^6 \) number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

---

**CICS considerations**
- If HEAP(,,ANYWHERE,,,) is in effect, the maximum size of a heap segment is 1 gigabyte (1024 MB).
- The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16 bytes CICS storage check zone. Without this accommodation, an extra page of storage is allocated when the storage allocation is below the 16 MB line.

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

**z/OS UNIX consideration**
In a multithreaded environment, user heap storage is shared by all threads in the process.
Usage notes

- Applications running in AMODE 24 that request heap storage get the storage below the 16M line regardless of the setting of ANYWHERE | BELOW.
- COBOL consideration—You can use the HEAP option to provide function similar to the VS COBOL II space management tuning table.
- PL/I consideration—For PL/I, the only case in which storage is allocated above the line is when all of the following conditions exist:
  - The user routine requesting the storage is running in 31-bit addressing mode.
  - HEAP(,,ANY,,,) is in effect.
  - The main routine runs in AMODE 31.
- PL/I MTF consideration—In a PL/I MTF application, HEAP specifies the heap storage allocation and management for a PL/I main task.

Performance consideration

You can improve performance with the HEAP run-time option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 68 for information on how to generate a report you can use to determine the optimum values for the HEAP run-time option.

For more information

- For more information about Language Environment heap storage, see z/OS Language Environment Programming Guide.
- For more information about the CEECRHP callable service, see "CEECRHP—Create new additional heap” on page 227.
- For more information about the CEEGTST callable service, see “CEEGTST—Get heap storage” on page 331.
- For more information about the RPTSTG run-time option, see “RPTSTG” on page 68.

HEAP64 (AMODE 64 only)

HEAP64 controls the allocation of user heap storage for AMODE 64 applications and specifies how that storage is managed. Heaps are storage areas containing user-controlled dynamically allocated variables or data. An example is C data allocated as a result of the malloc(), calloc(), and realloc() functions.

**Default** AMODE 64 **Value**

| AMODE 64 | HEAP64(1M,1M,KEEP,32K,32K,KEEP,4K,4K,FREE) |

**Syntax**

```
HEAP64 (init64, incr64, ..., KEEP, init31, incr31, ..., KEEP, init24, incr24, ..., KEEP, init24, incr24, FREE)
```
init64
Determines the initial allocation of heap storage obtained above the 2G bar.
Specify this value as nM bytes of storage. If a value of 0 or less is specified,
the default is used.

incr64
Determines the minimum size of any subsequent increment to the user
heap storage obtained above the 2G bar. Specify this value as nM bytes of
storage. If a value less than 0 is specified, the default is used.

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

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

init31
Determines the minimum initial size of user heap storage that is obtained
above the 16M line and below the 2G bar. This value can be specified as n, nK,
or nM number of bytes. If 0 is specified, the initial storage is obtained
on the first use and is based on the increment size. The amount of
allocated storage is rounded up to the nearest multiple of 8 bytes.

incr31
Determines the minimum size of any subsequent increment to user heap
storage that is obtained above the 16M line and below the 2G bar. This
value can be specified as n, nK, or nM number of bytes. The amount of
allocated storage is rounded up to the nearest multiple of 8 bytes.

init24
Determines the minimum initial size of user heap storage that is obtained
below the 16M line. This value can be specified as n, nK, or nM number of
bytes. If 0 is specified, the initial storage is obtained on the first use and is
based on the increment size. The amount of allocated storage is rounded
up to the nearest multiple of 8 bytes.

incr24
Determines the minimum size of any subsequent increment to user heap
storage that is obtained below the 16M line. This value can be specified as n, nK,
or nM number of bytes. The amount of allocated storage is rounded
up to the nearest multiple of 8 bytes.

**z/OS UNIX consideration**
In a multithreaded environment, user heap storage is shared by all threads the
process.

**Performance consideration**
You can improve performance with the HEAP64 run-time option by specifying
values that minimize the number of times the operating system allocates storage.
See “RPTSTG” on page 68 for information on how to generate a report you can use
to determine the optimum values for the HEAP64 run-time option.

**For more information**
- For more information about heap storage and heap storage tuning with storage
  report numbers, see z/OS Language Environment Programming Guide
- For more information about the RPTSTG run-time option, see “RPTSTG” on
  page 68
HEAPCHK

HEAPCHK performs diagnostic tests against the user heap.

Default Value

Non-CICS  HEAPCHK(OFF,1,0,0,0,1024,0,1024,0)
CICS       HEAPCHK(OFF,1,0,0,0,1024,0,1024,0)
AMODE 64   HEAPCHK(OFF,1,0,0,0,1024,0,1024,0)

Syntax

```plaintext
HEAPchk \{OFF|ON\},
\[frequency\], \[delay\], \[call depth\], \[pool call depth\].
\[num of entries\], \[pool number\].
\[num of entries 31\], \[pool number 31\].
```

**OFF**
Indicates that no heap checking or tracing is done regardless of the values of the remaining sub-options.

**ON**
Indicates that heap checking or tracing is activated based on the values of the remaining sub-options.

**frequency**
The frequency at which the user heap is checked for damage to the heap control information. It is specified as \( n, nK \) or \( nM \). A value of one (1) is the default and causes the heap to be checked at each call to a Language Environment heap storage management service. A value of \( n \) causes the user heap to be checked at every \( n \)th call to a Language Environment heap storage management service. A value of zero results in no check for damage to the user heap.

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

**call depth**
Specifies the depth of calls displayed in the traceback for the heap storage diagnostics report. A value of zero is the default that turns heap storage diagnostics reporting off. The heap storage diagnostics report consists of a set of tracebacks. Each traceback is a snapshot of the stack (to a specified call depth) for each request to obtain user heap storage that has not had a
corresponding free request. Use a value of 10 to ensure an adequate call depth is displayed so that you can identify the storage leak.

**pool call depth**
- Specifies the depth of calls in the traceback for each trace entry of a heap pools trace. A value of zero is the default that turns heap pools tracing off. The heap pools trace is an in-core wrapping trace. Each heap pool has a separate trace table. The heap pools trace is only formatted from a system dump using the IPCS verbexit LEDATA. Use a value of 10 to ensure an adequate call depth is displayed so that you can identify the storage leak.

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

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

**number of entries 31**
- Specifies the number of entries to be recorded in one heap pool trace table when an AMODE 64 application is using heap storage from 31-bit addressable storage (__malloc31()). Each pool has its own trace table. If the number of entries is 0, the heap pool trace table is not generated. This value is only supported in an AMODE64 environment.

**pool number 31**
- Specifies which pools are traced when an AMODE 64 application is using heap storage from 31-bit addressable storage (__malloc31()). You can trace either one pool or all pools. The value should be a valid pool number from 1 to 12. If the pool number is 0, all pools will be traced. This value is only supported in an AMODE64 environment.

**Usage notes**
- When user heap damage is detected, Language Environment immediately issues an ABEND U4042 RC=0. To obtain a system dump of this abend, either specify TDUMP for the third suboption of the DYNDUMP runtime option or ensure that a SYSMDUMP DD is available.
- If HEAPCHK(ON) is used with STORAGE(heap_free_value), all free areas of the heap are also checked.
- To request only a heap storage diagnostics report, you must specify a zero for frequency, a zero for pool call depth and a number n greater than zero for call depth. For example (HEAPCHK(ON,0,0,10,0)).
- Under normal termination conditions, if the call depth is greater than zero, the heap storage diagnostics report is written to the CEEDUMP report, independent of the TERMTHDACT setting.
- You can also generate a heap storage diagnostics report by calling CEE3DMP with the BLOCKS option.
- **Guideline:** Since HEAPCHK does not validate individual cells within a cell pool, you should specify HEAPPOLLS(OFF) when running HEAPCHK to diagnose storage overlays in the heap.
- To request heap pools tracing, set pool call depth to a non-zero value and number of entries (for AMODE 64 applications, number of entries, number of entries 31, or
HEAPCHK

both) to a non-zero value. To request only heap pools tracing, in addition, set
frequency to zero and call depth to zero. The heap pools trace is only formatted
from a system dump using the IPCS verbexit LEDATA.

- For AMODE 64 applications, number of entries and pool number control tracing for
the set of heap pools located in storage above the 2GB bar. Number of entries 31
and pool number 31 control tracing for the set of heap pools located in storage
above the 16MB line and below the 2GB bar. Pool call depth applies to both sets
of heap pools.

Performance consideration

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

For more information

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

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

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

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

Default Value
Non-CICS  HEAPPOOLS(OFF,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,
10,0,0,10,0,10,0,10,0,10,0,10,0,10)
CICS  HEAPPOOLS(OFF,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,
10,0,10,0,10,0,10,0,10,0,10,0)
AMODE 64  HEAPPOOLS(OFF,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,
10,0,10,0,10,0,10,0,10,0,10)

Syntax

HEAPPOOLS ( OFF, percentage )
ON cell-size
ALIGN (cell-size, pool-count)

OFF  Specifies that the heap pools algorithm is not used.
ON  Specifies that the heap pools algorithm is used.
ALIGN  Specifies that Language Environment will structure the storage for cells in
HEAPPOLLS

a heap pool so that a cell less than or equal to 248 bytes does not cross a cache line. For cells larger than 248 bytes, two cells never share a cache line.

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

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

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

Usage notes

- Cell pool sizes should be specified in ascending order.
- To use less than twelve heap pools, specify 0 for the cell size after the last heap pool to be used. For example if four heap pools are desired, use 0 for the fifth cell size when setting the HEAPPOLLS run-time option.
- If the percentage of the HEAP run-time option init size values does not allow for at least one cell, the system automatically adjusts the percentage to enable four cells to be allocated.
- The sum of the percentages may be more or less than 100 percent.
- Each heap pool is allocated as needed. The allocation of a heap pool can result in the allocation of a heap increment to satisfy the request.
- The FREE sub-option on the HEAP run-time option has no effect on any heap segment in which a heap pool resides. Each cell in a heap pool can be freed, but the heap pool itself is only released back to the system at enclave termination.
- To avoid wasting storage, see the heap pool tuning tips specified in z/OS Language Environment Programming Guide.
- The HEAPPOLLS run-time option has no effect when BELOW is specified as the location on the HEAP run-time option.
- Mixing of the storage management AWIs (CEEGTST(), CEEFRST() and CEECZST()) and the C/C++ intrinsic functions (malloc(), calloc(), realloc() and free() ) are not supported when operating on the same storage address. For example, if you request storage using CEEGTST(), then you may not use free() to release the storage.
- Using the ALIGN suboption might cause an increase in the amount of heap storage used by an application.
- You should examine the storage report and adjust storage tuning when first using the ALIGN suboption.
- The HEAPCHK run-time option does not validate individual heap pool cells.
- Use of a vendor heap manager (VHM) overrides the use of the HEAPPOLLS run-time option.
- If the RPTSTG run-time option is specified while using HEAPPOLLS, extra storage is obtained from the ANYHEAP and is used to complete the storage report on heappools. This extra storage is only allocated when both HEAPPOLLS and RPTSTG are used.
- The HEAPPOLLS run-time option can be used by AMODE 64 applications to manage user heap storage above the 16M line and below the 2G bar.
When cell-size is specified without parenthesis, pool-count defaults to 1 rather than being picked up from an earlier setting of pool-count. For example, specifying 128 is treated like specifying (128,1).

When cell-size is specified with parenthesis, pool-count must also be specified.

When pool-count is greater than 1, the size of each heap pool extent is determined by dividing the heap allocation for the cell-size by the pool-count.

Performance considerations

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

Use caution when using cells larger than 2K. Large gaps between cell sizes can lead to a considerable amount of storage waste. Properly tuning cell sizes with the help of RPTSTG is necessary to control the amount of virtual storage needed by the application.

When there are many successful get requests for the same size cell and the maximum elements used in the cell pool is high, this could be an indication that there is excessive contention allocating elements in the cell pool. Specifying pool-count greater than 1 might help relieve some of this contention. Multiple pools are allocated with the same cell size and a portion of the threads are assigned to allocate cells out of each of the pools.

HEAPPOOLS64 (C/C++ and AMODE 64 only)

<table>
<thead>
<tr>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEAP storage POOLS</td>
</tr>
</tbody>
</table>

The HEAPPOOLS64 run-time option is used to control an optional user heap storage management algorithm, known as heap pools, for AMODE 64 applications. This algorithm is designed to improve the performance of multithreaded C/C++ applications with a high frequency of calls to malloc(), calloc(), realloc(), free(), and operators new and delete. When active, heap pools virtually eliminates contention for user heap storage.

Default Value

AMODE 64

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

Syntax

```plaintext
HEAPPOOLS64 (OFF, cell-size, pool-count)
```

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF Specifies that the heappools algorithm is not being used.</td>
</tr>
<tr>
<td>ON Specifies that the heappools algorithm is being used.</td>
</tr>
<tr>
<td>ALIGN Specifies that Language Environment will structure the storage for cells in</td>
</tr>
</tbody>
</table>
HEAPPOOLS64

a heap pool so that a cell less than or equal to 240 bytes does not cross a
cache line. For cells larger than 240 bytes, two cells never share a cache
line.

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

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

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

Usage notes

- Cell pool sizes should be specified in ascending order.
- To use less than twelve heap pools, specify 0 for the cell size after the last heap
pool to be used. For example if four heap pools are desired, use 0 for the fifth
cell size when setting the HEAPPOOLS64 run-time option.
- Each heap pool is allocated as needed. The allocation of a heap pool can result
in the allocation of a heap increment to satisfy the request.
- Using the ALIGN suboption might cause an increase in the amount of heap
storage used by an application.
- Examine the storage report and adjust storage tuning when first using the
ALIGN suboption.
- The HEAPCHK run-time option does not validate individual heap pool cells.
- If you specify the RPTSTG run-time option while using HEAPPOOLS64, extra
storage is obtained from the LIBHEAP64 and is used to complete the storage
report on heappools. This extra storage is only allocated when both
HEAPPOOLS64 and RPTSTG are used.
- HEAPPOOLS run-time option can be used by AMODE 64 applications to
manage user heap storage above the 16M line and below the 2G bar.
- When cell-size is specified without parenthesis, pool-count defaults to 1 rather
than being picked up from an earlier setting of pool-count. For example,
specifying 128 is treated like specifying (128,1).
- When cell-size is specified with parenthesis, pool-count must also be specified.
- When pool-count is greater than 1, the size of each heap pool extent is
determined by dividing the heap allocation for the cell-size by the pool-count.

Performance considerations

- To improve the effectiveness of the heap pools algorithm, use the storage report
numbers generated by the RPTSTG run-time option as an aid in determining
optimum cell sizes and count.
- Use caution when using cells larger than 2K. Large gaps between cell sizes can
lead to a considerable amount of storage waste. Properly tuning cell sizes with
the help of RPTSTG is necessary to control the amount of virtual storage needed
by the application.

For more information

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

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

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

Default Value

<table>
<thead>
<tr>
<th>Environment</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>INFOMSGFILTER(OFF,...)</td>
</tr>
<tr>
<td>CICS</td>
<td>INFOMSGFILTER(OFF,...)</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>INFOMSGFILTER(OFF,...)</td>
</tr>
</tbody>
</table>

Syntax

```
INFOMsgfilter (OFF, ON, Foreground, Background)
```

- **OFF**  
  Turns off message filtering for all environments.

- **ON**  
  Turns on the message filtering for the specified environments.

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

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

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

INQPCOPN (Fortran only)

Derivation

INQuire the Pre-Connected units that are OPENed
INQPCOPN

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

**Default Value**
- **Non-CICS**: INQPCOPN
- **CICS**: INQPCOPN is ignored under CICS.

**Syntax**
```
INQpcopn
NOINQpcopn
```

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

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

**INTERRUPT**

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

**Default Value**
- **Non-CICS**: INTERRUPT(OFF)
- **CICS**: INTERRUPT is ignored under CICS.

**Syntax**
```
INTerrupt (OFF ON)
```

**OFF**
Specifies that Language Environment does not recognize attention interrupts.

**ON**
Specifies that Language Environment recognizes attention interrupts.

**z/OS UNIX consideration**
- In a multithreaded application, only one thread in the enclave is affected for a particular attention interrupt.
INTERRUPT

Usage notes

- PL/I consideration: Language Environment supports the PL/I method of polling code. Note that the PL/I routine must be compiled with the INTERRUPT compiler option in order for the INTERRUPT run-time option to have an effect.
- PL/I MTF consideration: To receive the attention interrupt, the PL/I program must be compiled with the INTERRUPT compiler option, and the INTERRUPT run-time option must be in effect.
- PL/I MTF consideration: The INTERRUPT option applies to the enclave. However, only one thread in the enclave is affected for a particular attention interrupt.
- If you have specified the TEST(ERROR) or TEST(ALL) run-time option, the interrupt causes the debug tool to gain control. See “TEST | NOTEST” on page 84 for more information about the TEST run-time option.

IOHEAP64 (AMODE 64 only)

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

Default Value

AMODE 64 IOHEAP64(1M,1M,FREE,12K,8K,FREE,4K,4K,FREE)

Syntax

```
IOHEAP64 (init64, incr64, KEEP, FREE)
init64
Determines the initial allocation of I/O heap storage obtained above the 2G bar. Specify this value as \( n \)M bytes of storage. If a value of 0 or less is specified, the default is used.

incr64
Determines the minimum size of any subsequent increment to the I/O heap storage obtained above the 2G bar. Specify this value as \( n \)M bytes of storage. If a value less than 0 is specified, the default is used.

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

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

init31
Determines the minimum initial size of I/O heap storage that is obtained above the 16M line and below the 2G bar. This value can be specified as \( n \),
nK, or nM number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

**incr31**
Determines the minimum size of any subsequent increment to I/O heap storage that is obtained above the 16M line and below the 2G bar. This value can be specified as n, nK, or nM number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

**init24**
Determines the minimum initial size of I/O heap storage that is obtained below the 16M line. This value can be specified as n, nK, or nM number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

**incr24**
Determines the minimum size of any subsequent increment to I/O heap storage that is obtained below the 16M line. This value can be specified as n, nK, or nM number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

**Performance consideration**
You can improve performance with the IOHEAP64 run-time option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 68 for information on how to generate a report you can use to determine the optimum values for the IOHEAP64 run-time option.

**For more information**
- For more information about heap storage and heap storage tuning with storage report numbers, see z/OS Language Environment Programming Guide.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 68.

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMODE 64</td>
<td>LIBHEAP64(1M,1M,FREE,16K,8K,FREE,8K,4K,FREE)</td>
</tr>
</tbody>
</table>
**LIBHEAP64**

- **Syntax**

  ```
  LIBHEAP64 (init64, incr64, KEEP, FREE), (init31, incr31, KEEP, FREE), (init24, incr24)
  ```

- **init64**
  Determines the initial allocation of library heap storage obtained above the 2G bar. Specify this value as \( n \)\( \text{M} \) bytes of storage. If a value of 0 or less is specified, the default is used.

- **incr64**
  Determines the minimum size of any subsequent increment to the library heap storage obtained above the 2G bar. Specify this value as \( n \)\( \text{M} \) bytes of storage. If a value less than 0 is specified, the default is used.

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

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

- **init31**
  Determines the minimum initial size of library heap storage that is obtained above the 16M line and below the 2G bar. This value can be specified as \( n \), \( n \text{K} \), or \( n \text{M} \) number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

- **incr31**
  Determines the minimum size of any subsequent increment to library heap storage that is obtained above the 16M line and below the 2G bar. This value can be specified as \( n \), \( n \text{K} \), or \( n \text{M} \) number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

- **init24**
  Determines the minimum initial size of library heap storage that is obtained below the 16M line. This value can be specified as \( n \), \( n \text{K} \), or \( n \text{M} \) number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

- **incr24**
  Determines the minimum size of any subsequent increment to library heap storage that is obtained below the 16M line. This value can be specified as \( n \), \( n \text{K} \), or \( n \text{M} \) number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.
Performance consideration

You can improve performance with the LIBHEAP64 run-time option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 68 for information on how to generate a report you can use to determine the optimum values for the LIBHEAP64 run-time option.

For more information

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

**CICS consideration**
The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16 byte CICS storage check zone. Without this accommodation, an extra page of storage is allocated.

**z/OS UNIX consideration**
The LIBSTACK option sets the library stack characteristics on each thread.

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

**Performance consideration**
You can improve performance with the LIBSTACK run-time option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 68 for information on how to generate a report you can use to determine the optimum values for the LIBSTACK run-time option.

**For more information**
- See "RPTSTG" on page 68 for more information about the RPTSTG run-time option.
- For more information about using the storage reports generated by the RPTSTG run-time option to tune the stacks, see z/OS Language Environment Programming Guide.

MSGFILE

**Derivation**
Message FILE

MSGFILE specifies the ddname and attributes of the data set (message file) where Language Environment directs the following output:
- All Language Environment messages
- Reports generated by the RPTOPTS and RPTSTG run-time options
- Output produced by the CEEMSG and CEEMOUT callable services

**Default Value**
Non-CICS  MSGFILE(SYSOUT,FBA,121,0,NOENQ)
CICS  MSGFILE is ignored under CICS.

**Syntax**

```
MSGFILE (ddname, recfm, lrecl, blksize, NOENQ, ENQ)
```
**MSGFILE**

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

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

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

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

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

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

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

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

**Usage notes**
- Under most circumstances, the NOENQ sub-option is sufficient and provides better performance. The ENQ sub-option is only needed when multiple Language Environment environments are running in the same address space and share the same message file.
- An instance when ENQ might be needed is a batch job that uses ATTACH to create sub-tasks. Each of the sub-tasks is potentially a distinct Language Environment environment, all running with the same default MSGFILE parameters. In this example, each of these environments shares the same message file.
To avoid conflicts while writing to the shared message file, use the ENQ sub-option. Using a different ddname for each environment can remove the need to use the ENQ sub-option.

- Compiler options, such as the COBOL OUTDD compiler option, can affect if your run-time output goes to MSGFILE ddname.
- If there is no blksize in the MSGFILE run-time option, in a data set definition, or in the label of an existing data set, the block size is determined as follows:
  - If recfm specifies unblocked fixed-length format records (F or FA) or undefined-format records (U or UA), blksize is the same as lrecl.
  - If recfm specifies unblocked variable-length format records (V or VA), blksize is lrecl plus 4.
  - If recfm specifies blocked records (FB, FBA, FBS, FBSA, VB, or VBA) for a DASD device on z/OS, Language Environment uses a blksize of 0 so the system can determine the optimum blksize.
  - If recfm specifies blocked fixed-length format records (FB, FBA, FBS, or FBSA) for a terminal, blksize is the same as lrecl.
  - If recfm specifies blocked variable-length format records (VB or VBA) for a terminal, blksize is lrecl plus 4.
  - For all other cases, blksize is calculated to provide the largest number of records per block, up to 100 records per block, that does not exceed the maximum blksize of 32760.

- Language Environment does not diagnose combinations of recfm, lrecl, and blksize that are not valid but the system can detect an error condition on the first attempt to write to the message file.
- Language Environment does not check the validity of the MSGFILE ddname. A ddname that is not valid generates an error condition on the first attempt to write to the message file.
- C/C++ consideration—C output directed to stderr and perror() messages go to the MSGFILE destination.
- PL/I consideration—Run-time messages in PL/I programs are directed to the message file instead of to the PL/I SYSPRINT STREAM PRINT file. User-specified output is directed to the PL/I SYSPRINT STREAM PRINT file. To direct this output to the message file, specify MSGFILE(SYSPRINT). Use of MSGFILE(SYSPRINT) restricts the LINESIZE for PL/I programs to a maximum of 255.
- Fortran consideration—To get the same message file function as with VS Fortran, specify MSGFILE(FTnnF001,UA,133) where nn is the unit number of the error unit. For more information, see Fortran Run-Time Migration Guide.
- Language Environment supports the use of a MSGFILE DDNAME dynamically allocated with the XTIOT, UCB nocapture, or DSAB-above-the-line options specified in the SVC99 parameters (S99TIOEX, S99ACUCB, S99DSABA flags).

For more information

- For more information about the RPTOPTS and RPTSTG run-time options, see “RPTOPTS” on page 66 and “RPTSTG” on page 68.
- For more information about the CEEMSG and CEEMOUT callable services, see “CEEMSG—Get, format, and dispatch a message” on page 402 and “CEEMOUT—Dispatch a message” on page 380.
- For details on how HLL compiler options affect messages, see information on HLL I/O statements and message handling in z/OS Language Environment Programming Guide.
For examples of getting and formatting messages, including HLL run-time output, see “CEEMSG—Get, format, and dispatch a message” on page 402.

For more information about perror() and stderr, see C message output information in z/OS Language Environment Programming Guide.

For more information about the CESE transient data queue, see z/OS Language Environment Programming Guide.

MSGQ

MSGQ specifies the number of instance specific information (ISI) blocks that Language Environment allocates on a per thread basis for use by the application. The ISI block contains information for Language Environment to use when identifying and reacting to conditions, providing access to q_data tokens, and assigning space for message inserts used with user-created messages. When an insufficient number of ISI blocks are available, Language Environment uses the least recently used ISI block.

Default Value

Non-CICS MSGQ(15)
CICS MSGQ is ignored under CICS.

Syntax

```plaintext
MSGQ (number)
```

number

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

Usage notes

- PL/I MTF consideration—In a PL/I MTF application, MSGQ sets the number of message queues allowed for each task.
- The CEECMI callable service allocates storage for ISI blocks if necessary. For information about using the CEECMI callable service, see “CEECMI—Store and load message insert data” on page 222.

For more information

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

NATLANG

NATLANG

Derivation

NATIONAL LANGUAGE
NATLANG

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

Default Value

| Non-CICS | NATLANG(ENU) |
| CICS     | NATLANG(ENU) |
| AMODE 64 | NATLANG(ENU) |

Syntax

```
/setlocale(ENU)
/setlocale(UEN)
/setlocale(JPN)
```

- **ENU** A 3-character ID specifying mixed-case U.S. English. Message text consists of SBCS characters and includes both uppercase and lowercase letters.
- **UEN** A 3-character ID specifying uppercase U.S. English. Message text consists of SBCS characters and includes only uppercase letters.
- **JPN** A 3-character ID specifying Japanese. Message text can contain a mixture of SBCS and DBCS characters.

Usage notes

- **Restriction:** CEE3LNG and CEESETL are not available to AMODE 64 applications.
- You can use the CEE3LNG callable service to set the national language.
- If you specify a national language that is not available on your system, Language Environment uses the IBM-supplied default ENU (mixed-case U.S. English).
- Language Environment is sensitive to the national language when it writes storage reports, option reports, and dump output.
  - When the national language is upper case U.S. English or Japanese, the environment variable _CEE_UPPERCASE_DATA can be used to determine if variable data in storage reports, options reports and dump output is in uppercase.
  - When this environment variable is set to YES, variable data (entry point names, program unit names, variable names, Trace Entry in EBCDIC data, hexadecimal/EBCDIC displays of storage) will be in uppercase.
  - When this environment variable is not set or set to a value other than YES, variable data will not be in uppercase. Variable data is never in uppercase when the national language is mixed-case U.S. English.
- Language Environment provides locales used in C/C++ to establish default formats for the locale-sensitive functions and locale callable services, such as date and time formatting, sorting, and currency symbols. To change the locale, you can use the setlocale() library function or the CEESETL callable service. The settings of CEESETL or setlocale() do not affect the setting of the NATLANG run-time option. NATLANG affects the Language Environment NLS
and date and time services. `setlocale()` and `CEESETL` affect only C/C++ locale-sensitive functions and Language Environment locale callable services. To ensure that all settings are correct for your country, use `NATLANG` and either `CEESETL` or `setlocale()`.

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

### For more information
- For more information about the `CEE3LNG` callable service, see "CEE3LNG—Set national language" on page 174.
- For more information about the `CEESETL` callable service, see "CEESETL—Set locale operating environment" on page 448.
- For more information on `setlocale()`, see `z/OS XL C/C++ Programming Guide`.

## OCSTATUS (Fortran only)

### Derivation

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Open Close STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCSTATUS</td>
<td></td>
</tr>
<tr>
<td>NOOCSTATUS</td>
<td></td>
</tr>
</tbody>
</table>

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

**Default Value**

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>OCSTATUS</td>
</tr>
<tr>
<td>CICS</td>
<td>OCSTATUS is ignored under CICS.</td>
</tr>
</tbody>
</table>

### Syntax

```
OCSTATUS
```

**OCSTATUS**

Specifies that file existence is checked with each `OPEN` statement to verify that the status of the file is consistent with `STATUS='OLD'` and `STATUS='NEW'`. It also specifies that file deletion occurs with each `CLOSE` statement with `STATUS='DELETE'` for those devices which support file deletion. Preconnected files are included in these verifications. OCSTATUS consistency checking applies to DASD files, PDS members, VSAM files, MVS labeled tape files, and dummy files only. For dummy files, the consistency checking occurs only if the file was previously opened successfully in the current program.

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

**NOOCSTATUS**

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

---

**Chapter 2. Using the Language Environment run-time options** 57
OCSTATUS

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

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

PAGEFRAMESIZE

Statement of Direction

IBM intends to extend the memory architecture for large page support (1 MB pages) to support pageable large pages. Externals and interfaces for extension of the memory architecture are being made available in z/OS V1.13 for early planning and development purposes only. This Statement of Direction is subject to change or withdrawal.

Derivation

PAGE FRAME SIZE

PAGEFRAMESIZE specifies the preferred page frame size in virtual storage for HEAP, ANYHEAP, and STACK storage obtained during application initialization and run-time.

Default Value

<table>
<thead>
<tr>
<th>Non-CICS</th>
<th>PAGEFRAMESIZE(4K,4K,4K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>PAGEFRAMESIZE is ignored under CICS.</td>
</tr>
</tbody>
</table>

Syntax

```
PAGeframesize ( heap_frame_size, anyheap_frame_size, stack_frame_size )
```

**heap_frame_size**

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

- **4K** Requests the default value of 4KB pages.
- **1M** Requests that 1MB large pages be used, if available.
anyheap_frame_size
  Specifies the preferred page frame size in virtual storage for initial
  anywhere heap storage allocation and any subsequent anywhere heap
  increments. The page frame size can be specified as one of the following
  values:
  4K  Requests the default value of 4KB pages.
  1M  Requests that 1MB large pages be used, if available.

stack_frame_size
  Specifies the preferred page frame size in virtual storage for initial stack
  storage allocation and any subsequent stack increments. The page frame
  size can be specified as one of the following values:
  4K  Requests the default value of 4KB pages.
  1M  Requests that 1MB large pages be used, if available.

Usage notes
  • You cannot set the PAGEFRAMESIZE during Language Environment installation
    (CEEDOPT, CEECOPT, or CEEROPT), in a CEEPRMxx parmlib member or with
    a SETCEE command.
  • You cannot specify PAGEFRAMESIZE with the CEEBXITA assembler user exit
    interface.
  • In an XPLINK environment, the stack_frame_size suboption only applies to the
    upward-growing stack.
  • If 1MB page frames are not available, the default 4KB page frame size will be
    used. No message is issued to indicate this behavior.
  • Page frame sizes larger than 4KB are not allowed below the 16MB line. If a
    PAGEFRAMESIZE parameter specifies 1MB but that storage type is allocated
    below the 16MB line, then the default 4KB page frames will be used. No
    message is issued to indicate this behavior; however, the run-time options report
    will show the value that was specified.
  • If any PAGEFRAMESIZE parameter specifies 1M, then all of the storage
    preallocated to the enclave will request 1MB page frames. The previous two
    usage notes apply as well.
  • By default, THREADSTACK storage comes from the library heap storage that is
    allocated with the ANYHEAP run-time option. To use 1MB page frames for the
    THREADSTACK, ensure the anyheap_frame_size suboption specifies 1M.
  • When running in a preinitialized environment with an @GETSTORE service
    routine, a flag is passed to indicate that storage was requested to be backed by
    1MB page frames. For more information about using 1MB page frames with an
    @GETSTORE service routine, see z/OS Language Environment Programming Guide.

Performance considerations
  Large pages are a special-purpose feature to improve performance; therefore, using
  large pages is not recommended for all types of workloads. For more information
  about large pages, see the following publications:
  • z/OS MVS Programming: Assembler Services Reference IAR-XCT
  • z/OS MVS Programming: Authorized Assembler Services Guide
  • z/OS MVS Programming: Assembler Services Guide
PC (Fortran only)

Derivation

Private Common blocks

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

Default Value

Non-CICS NOPC
CICS PC is ignored under CICS.

Syntax

NOPC
PC

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

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

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

PLIST (C only)

Derivation

Parameter LIST

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

- This option does not apply to non-C languages and can be specified only with the C #pragma runopts directive.
- You cannot set PLIST during Language Environment installation.

Default Value

Non-CICS PLIST(HOST)
CICS PLIST is ignored under CICS.
### Syntax

<table>
<thead>
<tr>
<th>Host</th>
<th>The parameter list is a character string. The string is located differently under various systems, as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS</td>
<td>If invoked by OSRUN, use the string presented in an MVS-like format located by the pointer held in R1. If not invoked by OSRUN, use the CMS extended parameter list.</td>
</tr>
<tr>
<td>TSO</td>
<td>If a command processor parameter list (CPPL) is detected, get the string from the command buffer. If a CPPL is not detected, assume a halfword-prefixed string in the MVS format.</td>
</tr>
<tr>
<td>MVS</td>
<td>Use the halfword-prefixed string.</td>
</tr>
<tr>
<td>CICS</td>
<td>The parameter list received by your C application is assumed to be in a CICS format.</td>
</tr>
<tr>
<td>CMS</td>
<td>The parameter list received by your C application is assumed to be in a CMS extended parameter list format.</td>
</tr>
<tr>
<td>IMS</td>
<td>The parameter list received by your C application is assumed to be in an IMS format.</td>
</tr>
<tr>
<td>MVS</td>
<td>The parameter list received by your C application is assumed to be in an MVS format.</td>
</tr>
<tr>
<td>OS</td>
<td>The parameter list received by your C application is assumed to be in an OS style.</td>
</tr>
<tr>
<td>TSO</td>
<td>The parameter list received by your C application is assumed to be in a CPPL format.</td>
</tr>
</tbody>
</table>

### Usage notes

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

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

**Default Value**

- **Non-CICS**: PLITASKCOUNT(20)
- **CICS**: PLITASKCOUNT is ignored under CICS.

**Syntax**

\[
\text{PLITaskcount (~tasks~)}
\]

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

**Usage notes**

- A value of zero (0) assumes the IBM-supplied default of 20.
- If tasks or the IBM-supplied default of 20 exceeds the z/OS UNIX installation default of the maximum number of threads, Language Environment assumes the z/OS UNIX installation default.
- If a request to create a task would put the number of currently active tasks over the allowable limit, condition IBM0566S is signalled and the task is not created.

**POSIX**

**Derivation**

Portable Operating System Interface - X

POSIX specifies if the enclave can run with the POSIX semantics. POSIX is an application characteristic that is maintained at the enclave level. After you have established the characteristic during enclave initialization, you cannot change it.

**Default Value**

- **Non-CICS**: POSIX(OFF)
- **CICS**: POSIX is ignored under CICS.
- **AMODE 64**: POSIX(ON)
Usage notes

- When you set POSIX to ON, you can use functions that are unique to POSIX, such as pthread_create().
- POSIX(ON) applies to z/OS but explicitly excludes CICS. If you set POSIX to ON while an application is running under CICS, you receive a warning message, POSIX is set OFF, and the application continues to run. You can specify POSIX(ON) for both DB2® and IMS applications.
- When you set POSIX to ON while an application is running under CICS, you receive a warning message, POSIX is set OFF, and the application continues to run.
- One of the effects of POSIX(ON) is the enablement of POSIX signal handling semantics, which interact closely with the Language Environment condition handling semantics.
- ANSI C programs can access the z/OS UNIX Hierarchical File System (HFS) on MVS independent of the POSIX setting. Where ambiguities exist between ANSI and POSIX semantics, the POSIX run-time option setting indicates the POSIX semantics to follow.
- Within nested enclaves, only one enclave can have the POSIX option set to ON. All other nested enclaves must have the POSIX option set to OFF. When a second nested enclave tries to specify the run-time option POSIX(ON) within one Language Environment process, Language Environment ends with abend U4093, reason code 172.

For more information

For more information on POSIX functions that have a kernel dependency or a POSIX ON dependency, see z/OS XL C/C++ Run-Time Library Reference.

PROFILE

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

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

Default Value
Non-CICS PROFILE(OFF,"")
CICS PROFILE(OFF,"")
AMODE 64 PROFILE(OFF,"")
PROFILE

Syntax

```
PROFILE (OFF, string)
```

**OFF** Indicates that the profile facility is inactive.

**ON** Indicates that the profile facility is active.

**string** Profile options that Language Environment passes to the profiler installed. You can enclose the string in either single or double quotation marks. The maximum length of the string is 250 bytes when specified on program invocation or from a compiler directive.

For more information

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

PRTUNIT (Fortran only)

**Derivation**

```
PRinT UNIT
```

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

**Default Value**

- **Non-CICS**: PRTUNIT(6)
- **CICS**: PRTUNIT is ignored under CICS.

**Syntax**

```
PRTunit (number)
```

**number** A valid unit number in the range 0-99. You can establish your own default number at installation time.

PUNUNIT (Fortran only)

**Derivation**

```
PUNch UNIT
```


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

**Default Value**

- **Non-CICS**: PUNUNIT(7)
- **CICS**: PUNUNIT is ignored under CICS.

**Syntax**

```
>RUNUNIT (number)
```

`number`

A valid unit number in the range 0-99. You can establish your own default number at installation time.

---

**RDRUNIT (Fortran only)**

**Derivation**

```
RDRUNIT
```

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

**Default Value**

- **Non-CICS**: RDRUNIT(5)
- **CICS**: RDRUNIT is ignored under CICS.

**Syntax**

```
>RREADR (number)
```

`number`

A valid unit number in the range 0-99. You can establish your own default number at installation time.

---

**RECPAD (Fortran only)**

**Derivation**

```
RECPAD
```

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

**Default Value**

- **Non-CICS**: NORECPAD
- **CICS**: RECPAD is ignored under CICS.
RECPAD

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

REDIR | NOREDIR (C Only)

Derivation
REDIREction

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

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

Default Value
- Non-CICS: REDIR
- CICS: REDIR is ignored under CICS.
- AMODE 64: REDIR

Syntax

REDIR | NOREDIR

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

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

For more information
See “ARGPARSE | NOARGPARSE (C only)” on page 15 for a description of ARGPARSE.

RPTOPTS

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

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>RPTOPTS(OFF)</td>
</tr>
<tr>
<td>CICS</td>
<td>RPTOPTS(OFF)</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>RPTOPTS(OFF)</td>
</tr>
</tbody>
</table>

**Syntax**

```
RPTopts (OFF ON)
```

**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.

**Usage notes**

- For AMODE 64 applications, Language Environment writes the options report to stderr.
- In some cases, RPTOPTS will not generate the options report if your application ends abnormally.
- In a non-CICS environment, Language Environment directs the report to the ddname specified in the MSGFILE run-time option. Under CICS, with RPTOPTS(ON), Language Environment writes the options report to the CESE queue when the transaction ends successfully.
- If the RPTSTG run-time option is specified while using HEAPPOOLS, extra storage is obtained from the ANYHEAP and is used to complete the storage report on heappools. This extra storage is only allocated when both HEAPPOOLS and RPTSTG are used.

**Performance consideration**

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

**For more information**

- See "MSGFILE" on page 52 for more information about the MSGFILE run-time option.
- For an example and complete description of the options report, see z/OS Language Environment Debugging Guide.
RPTSTG generates, after an application has run, a report of the storage the application used. Language Environment writes storage reports only in mixed-case U.S. English.

You can use the storage report information to help you set the ANYHEAP, BELOWHEAP, HEAP, HEAP64, HEAPPOLLS, HEAPPOLLS64, IOHEAP64, LIBHEAP64, LIBSTACK, STACK, STACK64, THREADHEAP, THREADSTACK, and THREADSTACK64 run-time options for the best storage tuning. For an example and complete description of the storage report, see z/OS Language Environment Debugging Guide.

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Non-CICS</th>
<th>RPTSTG(OFF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>RPTSTG(OFF)</td>
<td></td>
</tr>
<tr>
<td>AMODE 64</td>
<td>RPTSTG(OFF)</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```
RPTSTG (OFF | ON)
```

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

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

**CICS consideration**

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

**z/OS UNIX consideration**

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

**Usage notes**

- For AMODE 64 applications, Language Environment writes the storage report to stderr.
- When a vendor heap manager (VHM) is active, the Language Environment Storage Report will have a text line indicating that the user heap for C/C++ part of the enclave is managed separately. The VHM is expected to write its own storage report to the stderr stream.
- In some cases, RPTSTG will not generate the storage report if your application ends abnormally.
RPTSTG

- RPTSTG includes PL/I task-level information on stack and heap usage.
- The phrases “Number of segments allocated” and “Number of segments freed” represent the number of GETMAIN and FREEMAIN requests, respectively.

Performance consideration

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

For more information

- For more information about tuning your application with storage numbers, see z/OS Language Environment Programming Guide.
- For more information about the MSGFILE run-time option, see “MSGFILE” on page 52.
- For an example and complete description of the storage report, see z/OS Language Environment Debugging Guide.

RTEREUS (COBOL only)

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Run Time Environment REUSE</th>
</tr>
</thead>
</table>

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, CEEUOPT, CEEROPT or the assembler user exit.

Default Value

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>NORTEREUS</td>
</tr>
<tr>
<td>CICS</td>
<td>RTEREUS is ignored under CICS.</td>
</tr>
</tbody>
</table>

Syntax

```
NORTEREUS
RTEREUS
```

NORTEREUS

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

RTEREUS

- Initializes the run-time environment to be reusable when the first COBOL program is invoked.
Usage notes

- **Restriction:** Enterprise COBOL programs compiled with the THREAD compiler option does not run with RTEREUS(ON).
- **Guideline:** Avoid using RTEREUS(ON) as an installation default. If you do use RTEREUS, use it for specific applications only.
- Under Language Environment, RTEREUS(ON) is only supported in a single enclave environment unless you modify the behavior using the IGZERREO CSECT. With the IBM supplied default setting for COBOL's reusable environment, applications that create multiple enclaves will terminate with error message IGZ0168S. Multiple enclaves can be created by applications that use SVC LINK or CMSCALL to invoke application programs. One example is when an SVC LINK is used to invoke an application program under ISPF that is using ISPF services (such as CALL 'ISPLINK' and ISPF SELECT).
- If a Language Environment reusable environment is established (using RTEREUS), any attempts to run a C or PL/I main program under Language Environment will fail. For example, when running on ISPF with RTEREUS(ON):
  - The first program invoked by ISPF is a COBOL program. A Language Environment reusable environment is established.
  - At some other point, ISPF invokes a PL/I or C program. The initialization of the PL/I or C program fails.
- If a large number of COBOL programs run (using RTEREUS) under the same MVS task, you can encounter out-of-region abends. This is because all storage acquired by Language Environment to run COBOL programs is kept in storage until the MVS task ends or the Language Environment environment is terminated.
- Language Environment storage and run-time options reports are not produced by Language Environment (using RTEREUS) unless a STOP RUN is issued to end the enclave.
- IMS consideration—RTEREUS is not recommended for use under IMS.
- The IGZERREO CSECT affects the handling of program checks in the non-Language Environment-enabled driver that repeatedly invokes COBOL programs. It also affects the behavior of running COBOL programs in a nested enclave when a reusable environment is active.

Performance consideration

- You must change STOP RUN statements to GOBACK statements to gain the benefits of RTEREUS. STOP RUN terminates the reusable environment. If you specify RTEREUS and use STOP RUN, Language Environment recreates the reusable environment on the next invocation of COBOL. Doing this repeatedly degrades performance, because a reusable environment takes longer to create than does a normal environment.
- The IGZERREO CSECT affects the performance of running with RTEREUS.
- Language Environment also offers preinitialization support in addition to RTEREUS.

For more information

- For more information about CEEUOPT, CEEROPT or CEEDOPT, see z/OS Language Environment Customization.
- For more information about IGZERREO, see z/OS Language Environment Customization.
SIMVRD (COBOL only)

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

**Default Value**
- **Non-CICS**: NOSIMVRD
- **CICS**: SIMVRD is ignored under CICS.

**Syntax**
```
NOSIMVRD
SIMVRD
```

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

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

**For more information**
For more details, see one of the following publications:
- *Enterprise COBOL for z/OS, V4R2, Programming Guide* SC23-8529
- *Enterprise COBOL for z/OS, V3R4, Programming Guide* SC27-1412
- *COBOL for OS/390 & VM Programming Guide*

STACK

STACK controls the allocation of the thread’s stack storage for both the upward and downward-growing stacks. Typical items residing in the upward-growing stack are C or PL/I automatic variables, COBOL LOCAL-STORAGE data items, and work areas for run-time library routines. The downward-growing stack is allocated only in an XPLINK environment.

**Default Value**
- **Non-CICS**: STACK(128K,128K,ANYWHERE,KEEP,512K,128K)
- **CICS**: STACK(4K,4080,ANYWHERE,KEEP,4K,4080)
**STACK**

**Syntax**

```
STACK (usinit_size, usincr_size, ANYWHERE, KEEP, FREE, dsinit_size, dsincr_size)
```

*usinit_size*

Determines the initial allocation of the upward-growing stack storage. This value can be specified as $n$, $nK$, or $nM$ bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

*usinit_size* can be preceded by a minus sign. In environments other than CICS, if you specify a negative number Language Environment uses all available storage minus the amount specified for the initial stack storage.

A size of "0" or "−0" requests half of the largest block of contiguous storage in the region below the 16 MB line.

*usincr_size*

Determines the minimum size of any subsequent increment to the upward-growing stack storage. This value can be specified as $n$, $nK$, or $nM$ bytes of storage. The actual amount of allocated storage is the larger of two values—*usincr_size* or the requested size—rounded up to the nearest multiple of 8 bytes.

If you specify *usincr_size* as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 8 bytes, is obtained.

The requested size is the amount of storage a routine needs for a stack frame.

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

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

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

**ANYWHERE | ANY**

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

**BELOW**

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

**KEEP**

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

dsinit_size Determines the initial allocation of the downward-growing stack storage. This value can be specified as n, nK, or nM bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 16 bytes.

dsincr_size Determines the minimum size of any subsequent increment to the downward-growing stack storage. This value can be specified as n, nK, or nM bytes of storage. The actual amount of allocated storage is the larger of two values—dsincr_size or the requested size—rounded up to the nearest multiple of 16 bytes.

CICS consideration
- dsinit_size and dsincr_size sub-options are ignored under CICS.
- The maximum initial and increment size for CICS above 16 MB is 1 gigabyte (1024 MB).
- The minimum for the initial size is 4K.
- STACK(0), STACK (-0), and STACK (-n) are all interpreted as STACK(4K) under CICS.
- The default increment size under CICS is 4080 bytes, rather than 4096 bytes, to accommodate the 16 bytes CICS storage check zone. Without this accommodation, an extra page of storage is allocated when the storage allocation is below the 16 MB line.

z/OS UNIX consideration
- The STACK option specifies the characteristics of the user stack for the initial thread. In particular, it gets the initial size of the user stack for the initial thread. The characteristics that indicate incr_size, ANYWHERE, and KEEP | FREE apply to any thread created using pthread_create. Language Environment gets the initial stack size from the thread’s attribute object specified in the pthread_create function. The default size to be set in the thread’s attribute object is obtained from the STACK run-time option's initial size.
- Guideline: The default setting for STACK under z/OS UNIX is STACK=(12K,12K,ANYWHERE,KEEP,512K,128K).

Usage notes
- Applications running with ALL31(OFF) must specify STACK(_,BELOW,) to ensure that stack storage is addressable by the application.
- When an application is running in an XPLINK environment, the STACK run-time option is forced to STACK(_,ANY,). Only the third suboption of the STACK run-time option is changed by this action, to indicate that stack storage can be allocated anywhere in storage. No message will be issued to indicate this action. In this case, if a Language Environment run-time options report is generated using the RPTOPTS run-time option, the STACK option will be reported as "Override" under the LAST WHERE SET column.
- If the initial routine of the Language Environment application is AMODE 24, the STACK run-time option will be forced to STACK(_,BELOW). Only the third suboption of the STACK run-time option is changed by this action. No message will issued to indicate this action. However, if a Language Environment run-time
options report is generated using the RPTOPTS run-time option, the STACK option will be reported as "Override" under the LAST WHERE SET column.

- The *dsinit_size* and *dsincr_size* values are not the actual amounts of storage obtained. The actual size of the storage obtained is 4K larger (8K if a 4K page alignment cannot be guaranteed) to accommodate the guard page.

- PL/I consideration—PL/I automatic storage above the 16 MB line is supported under control of the Language Environment STACK option. When the Language Environment stack is above, PL/I temporaries (dummy arguments) and parameter lists (for reentrant/recursive blocks) also reside above.

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

If an OS PL/I application does not contain any edited stream I/O and if it is running with AMODE 31, you can relink it with Language Environment to use STACK(_ANY,). Doing so is particularly useful under CICS to help relieve below-the-line storage constraints.

- PL/I MTF consideration—The STACK option allocates and manages stack storage for the PL/I main task only.

### Performance consideration

You can improve performance with the STACK run-time option by specifying values that minimize the number of times the operating system allocates storage. See "RPTSTG" on page 68 for information on how to generate a report you can use to determine the optimum values for the STACK run-time option.

### For more information

- See "ALL31" on page 12 for more information about the ALL31 run-time option.
- See "RPTSTG" on page 68 for more information about the RPTSTG run-time option.
- For more information about using the storage reports generated by the RPTSTG run-time option to tune the stacks, see z/OS Language Environment Debugging Guide.
- See "XPLINK" on page 98 for more information about the XPLINK run-time option.

### STACK64 (AMODE 64 only)

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

**Default Value**

<table>
<thead>
<tr>
<th>AMODE 64</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STACK64(1M,1M,128M)</td>
</tr>
</tbody>
</table>

**Syntax**

```
STACK64( initial , increment , maximum )
```
initial
Determine the size of the initial stack segment. The storage is contiguous. This value is specified as \( nM \) bytes of storage.

increment
Determine the minimum size of any subsequent increment to the downward-growing stack area. This value is specified as \( nM \) bytes of storage. The actual amount of allocated storage is the larger of two values—increment or the requested size—rounded up to the nearest 1MB. If you specify increment as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 1MB, is obtained. The requested size is the amount of storage a routine needs for a stack frame.

maximum
Specifies the maximum stack size. This value is specified as \( nM \) bytes of storage. When the maximum size is less than the initial size, initial is used as the maximum stack size.

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

Performance consideration
You can improve performance with the STACK64 run-time option by specifying values that minimize the number of times the operating system allocates storage. See “RPTSTG” on page 68 for information on how to generate a report you can use to determine the optimum values for the STACK64 run-time option.

For more information
- See “RPTSTG” on page 68 for more information about the RPTSTG run-time option.
- For more information about heap storage and heap storage tuning with storage report numbers, see z/OS Language Environment Programming Guide.
- For more information about using the storage reports generated by the RPTSTG run-time option to tune the stacks, see z/OS Language Environment Debugging Guide.

STORAGE
STORAGE controls the initial content of storage when allocated and freed. It also controls the amount of storage that is reserved for the out-of-storage condition. If you specify one of the parameters in the STORAGE run-time option, all allocated storage processed by that parameter is initialized to the specified value. Otherwise, it is left uninitialized. You can use the STORAGE option to identify uninitialized...
STORAGE

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.

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>STORAGE(NONE,NONE,NONE,0K)</td>
</tr>
<tr>
<td>CICS</td>
<td>STORAGE(NONE,NONE,NONE,0K)</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>STORAGE(NONE,NONE,NONE,)</td>
</tr>
</tbody>
</table>

Syntax

```plaintext
STORAGE(heap_alloc_value,heap_free_value,dsa_alloc_value,reserve_size)
```

heap_alloc_value

The initialized value of any heap storage allocated by the storage manager. You can specify `heap_alloc_value` as:

- A single character enclosed in 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 **NONE**, the allocated heap storage is not initialized.

heap_free_value

The value of any heap storage freed by the storage manager is overwritten. You can specify `heap_free_value` as:

- A single character enclosed in quotes. For example, a `heap_free_value` of 'f' overwrites freed heap storage to `X'868686...86'; 'B' overwrites freed heap storage to `X'C2C2C2...C2'`.
- Two hex digits without quotes. A `heap_free_value` of FE overwrites freed heap storage with `X'FEFEFE...FE'`.
- **NONE** If you specify **NONE**, the freed heap storage is not initialized.

dsa_alloc_value

The initialized value of stack frames from the Language Environment stack. A stack frame is dynamically-acquired storage that is composed of a standard register save area and the area available for automatic storage. The dsa_alloc_value has no effect on the XPLINK or AMODE 64 downwards growing stack.

If specified, all Language Environment stack storage, including automatic variable storage, is initialized to dsa_alloc_value. Stack frames allocated outside the Language Environment stack are never initialized.

You can specify dsa_alloc_value as:

- A single character enclosed in 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
STORAGE

'A' as the dsa_alloc_value, stack storage is initialized to X'C1C1C1...C1'. A dsa_alloc_value of 'F' initializes stack storage to X'C6C6C6...C6', 'd' to X'848484...84'.

- Two hex digits without 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'FEFEFE...FE'. A dsa_alloc_value of 00 initializes stack storage to X'00', FF to X'FFFFFF...FF'.
- CLEAR If you specify CLEAR, any unused portion of the initial upward growing stack segment is initialized to binary zeroes, just before the main procedure gains control. This value has no effect on any stack increments or on the XPLINK or AMODE 64 downward growing stack.
- NONE If you specify NONE, the stack storage is not initialized.

reserve_size
The amount of storage for the Language Environment storage manager to reserve in the event of an out-of-storage condition. You can specify the reserve_size value as n, nK, or nM bytes of storage. The amount of storage is rounded to the nearest multiple of 8 bytes.

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

If you specify reserve_size as 0, no reserve segment is allocated. The default reserve_size is 0, so no reserve segment is allocated. If you do not specify a reserve segment and your application exhausts storage, the application terminates with abend 4088 and a reason code of 1024.

If you specify a reserve_size that is greater than 0 on a non-CICS system, Language Environment does not immediately abend when your application runs out of storage. Instead, when the stack overflows, Language Environment uses the reserve stack as the new segment and signals a CEE0PD out of storage condition. This allows a user-written condition handler to gain control for this signal and release storage. If the reserve stack segment overflows while this is happening, Language Environment terminates with abend 4088 and reason code of 1004. The reserve stack segment is not freed until thread termination. It is acquired from 31-bit storage if the STACK(,,ANY,,,) runtime option is set or 24-bit storage when STACK(,,BELOW,,,) is requested. If a determination is made to activate the reserve stack, the reserve size should be set to a minimum of 32K to support Language Environment condition handling and messaging internal routines as well as the user condition handler. When using the reserve stack in a multithreaded environment, it is recommended that the ALL31(ON) and STACK(,,ANY,,,) options also be in effect.

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

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

z/OS UNIX consideration
A reserve stack of the size specified by the reserve_size suboption of STORAGE is allocated for each thread.
STORAGE

Usage notes

- The 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('**')

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

Performance considerations

- Use STORAGE(NONE,NONE,NONE) when you are not debugging.
- Using STORAGE to control initial values can increase program 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.
- You should not use STORAGE(,,00) in any performance-critical application.

TERMTHDACT

Derivation

TERMinating THreaD ACTions

TERMTHDACT sets the level of information that is produced when Language Environment percolates a condition of severity 2 or greater beyond the first routine's stack frame. The Language Environment service CEE3DMP is called for the TRACE, UATRACE, DUMP, and UADUMP suboptions of TERMTHDACT.

The following CEE3DMP options are passed for TRACE:

- NOENTRY
- CONDITION
- TRACEBACK
- THREAD(ALL)
- NOBLOCK
- NOSTORAGE
- VARIABLES
- NOFILES
- STACKFRAME(ALL)
- PAGESIZE(60)
- FNAME(CEEDUMP)
- GENOPTS

The following options are passed for DUMP and UADUMP:
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.

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>TERMTHDACT(TRACE,CESE,96)</td>
</tr>
<tr>
<td>CICS</td>
<td>TERMTHDACT(TRACE,CESE,96)</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>TERMTHDACT(TRACE,,96)</td>
</tr>
</tbody>
</table>

**Syntax**

```
TERMTHDACT (TRACE, QUIET, MSG, DUMP, UATRACE, CICSDDS, reg_stor_amount)
```

**TRACE**

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

**QUIET**

Specifies that Language Environment does not generate a message when a thread terminates due to an unhandled condition of severity 2 or greater.

**MSG**

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination.

**DUMP**

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

**UATRACE**

Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination, a trace of the active routines on the activation stack, and a U4039 system dump of the user address space.
TERMTHDACT

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

UAADVMP
Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination, a trace of the active routines on the activation stack, a Language Environment dump, and a U4039 system dump of the user address space. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space. Under CICS, you will get a CICS transaction dump.

UAONLY
Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a U4039 system dump of the user address space. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space. Under CICS, you will get a CICS transaction dump.

UAIMM
Specifies to Language Environment that prior to condition management processing, for abends and program interrupts that are conditions of severity 2 or greater, Language Environment will immediately request the operating system to generate a system dump of the original abend/program interrupt of the user address space. Due to an unhandled condition of severity 2 or greater, Language Environment generates a U4039 system dump of the user address space. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space. After the dump is taken by the operating system, Language Environment condition manager can continue processing. If the thread terminates due to an unhandled condition of Severity 2 or higher, then Language Environment will terminate as if TERMTHDACT(QUIET) was specified.

Note: For software-raised conditions or signals, UAIMM behaves the same as UAONLY.

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

CICSDDS
Specifies that Language Environment dump output will be written to the new CICS transaction dump that contains both CICS and CEEDUMP data. This option is ignored in a 64-bit environment.

reg_stor_amount
Controls the amount of storage to be dumped around registers. This amount can be in the range from 0 to 256 bytes. The amount specified will be rounded up to the nearest multiple of 32. The default amount is 96 bytes.

CICS consideration

All TERMTHDACT output is written to the data queue based on the setting of CESE or CICSDDS. See Table 5 on page 81 for a summary of the results of the different options that are available.
Table 5. Condition Handling of 0Cx ABENDS

<table>
<thead>
<tr>
<th>Options</th>
<th>TERMTHDACT(option,CESE,)</th>
<th>TERMTHDACT(option,CICSDDS,)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIET</td>
<td>• No output.</td>
<td>• No output.</td>
</tr>
<tr>
<td></td>
<td>• ASRA or user ABEND issued.</td>
<td>• ASRA or user ABEND issued.</td>
</tr>
<tr>
<td>MSG</td>
<td>• Message written to CESE queue or MSGFILE.</td>
<td>• Message written to CESE queue or MSGFILE.</td>
</tr>
<tr>
<td></td>
<td>• ASRA or user ABEND issued.</td>
<td>• ASRA or user ABEND issued.</td>
</tr>
<tr>
<td>TRACE</td>
<td>• Message written to CESE queue.</td>
<td>• Message written to CESE or MSGFILE.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td>• Traceback included in CICS transaction dump for this ABEND.</td>
</tr>
<tr>
<td></td>
<td>• ASRA or user ABEND issued.</td>
<td>• ASRA or user ABEND issued.</td>
</tr>
<tr>
<td>DUMP</td>
<td>• Message written to CESE queue.</td>
<td>• Invalid sub-option combination. Not supported.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CEEDUMP to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ASRA or user ABEND issued.</td>
<td></td>
</tr>
<tr>
<td>UATRACE</td>
<td>• Message written to CESE queue.</td>
<td>• Message written to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td>• Traceback included in CICS transaction dump for this ABEND.</td>
</tr>
<tr>
<td></td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td>• U4039 transaction dump in CICS dump data set.</td>
</tr>
<tr>
<td></td>
<td>• ASRA or user ABEND issued.</td>
<td>• ASRA or user ABEND issued.</td>
</tr>
<tr>
<td>UADUMP</td>
<td>• Message written to CESE queue.</td>
<td>• Invalid sub-option combination. Not supported.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CEEDUMP written to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ASRA or user ABEND issued.</td>
<td></td>
</tr>
<tr>
<td>UAONLY</td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td>• No changes in behavior for CICSDDS.</td>
</tr>
<tr>
<td>UAIMM</td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td>• No changes in behavior for CICSDDS.</td>
</tr>
</tbody>
</table>

Note: Program checks end in ASRx (most commonly ASRA) CICS abend with a CICS dump in the dump data set. Abends end with the abend code provided on the EXEC CICS ABEND command with a CICS dump in the dump data set if the NODUMP option was NOT specified.

Table 6. Handling of Software Raised Conditions

<table>
<thead>
<tr>
<th>Options</th>
<th>TERMTHDACT(option,CESE,)</th>
<th>TERMTHDACT(option,CICSDDS,)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIET</td>
<td>• No output.</td>
<td>• No output.</td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued with CANCEL and NODUMP options.</td>
<td>• U4038 abend issued with CANCEL and NODUMP options.</td>
</tr>
<tr>
<td>MSG</td>
<td>• Message written to CESE queue or MSGFILE.</td>
<td>• Message written to CESE queue or MSGFILE.</td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued.</td>
<td>• U4038 abend issued.</td>
</tr>
<tr>
<td>TRACE</td>
<td>• Message written to CESE queue or MSGFILE.</td>
<td>• Message written to CESE queue or MSGFILE.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td>• Traceback written to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued.</td>
<td>• U4038 abend issued.</td>
</tr>
</tbody>
</table>
Table 6. Handling of Software Raised Conditions (continued)

<table>
<thead>
<tr>
<th>Options</th>
<th>TERMTHDACT(option,CESE,)</th>
<th>TERMTHDACT(option,CICSDDS,)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Message written to CESE queue or MSGFILE.</td>
<td>• Invalid sub-option combination. Not supported.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CEEEDUMP written to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued.</td>
<td></td>
</tr>
<tr>
<td>UATRACE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Message written to CESE queue or MSGFILE.</td>
<td>• Message written to CESE queue or MSGFILE.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td>• Traceback written to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td>• U4039 transaction dump in CICS dump data set.</td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued.</td>
<td>• U4038 abend issued.</td>
</tr>
<tr>
<td>UADUMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Message written to CESE queue or MSGFILE.</td>
<td>• Invalid sub-option combination. Not supported.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CEEEDUMP written to CESE queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued.</td>
<td></td>
</tr>
<tr>
<td>UAONLY</td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td>• No changes in behavior for CICSDDS.</td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued.</td>
<td></td>
</tr>
<tr>
<td>UAIMM</td>
<td>• U4039 transaction dump in CICS dump data set.</td>
<td>• Incorrect sub-option combination. Not supported.</td>
</tr>
<tr>
<td></td>
<td>• U4038 abend issued.</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

1. CICS is told about software raised error conditions for DUMP and TRACE.
2. When assembling a CEECOPT, CEEROPT, or CEEUOPT, the CICSDDS option cannot be issued with DUMP or UADUMP. This results in a RC=8, CEEEXOPT issues the following message, and the setting is forced to TRACE:
   
   8, The TERMTHDACT level setting of DUMP
   8, conflicts with the CICSDDS suboption.
   8, A level of TRACE or less must be used with CICSDDS.
   8, The TERMTHDACT level suboption
   8, was set to TRACE.

3. Language Environment requests a CICS transaction dump via the U4039 abend.
5. Running with something like TERMTHDACT(TRACE,CICSDDS) in the CEECOPT or CEEROPT and then creating a CEEUOPT without specifying the second operand (for example, TERMTHDACT(DUMP)) results in the CICS dump data set as the output destination. Also, the following message occurs in the CESE queue and the traceback is written to the CICS transaction dump data set.
Usage notes

- A run-time options report will be generated and placed at the end of the enclave information whenever the TRACE, UATRACE, DUMP and UADUMP options are invoked.
- PL/I considerations—After a normal return from a PL/I ERROR ON-unit or from a PL/I FINISH ON-unit, Language Environment considers the condition unhandled. If a GOTO is not performed and the resume cursor is not moved, the thread terminates. The TERMTHDACT setting guides the amount of information that is produced. The message is not presented twice.
- PL/I MTF considerations—
  - TERMTHDACT applies to a task when the task terminates abnormally due to an unhandled condition of severity 2 or higher that is percolated beyond the initial routine’s stack frame.
  - When a task ends with a normal return from an ERROR ON-unit and other tasks are still active, a dump is not produced even when the TERMTHDACT option DUMP, UADUMP, UAONLY, or UAIMM is specified.
  - All active subtasks created from the incurring task also terminate abnormally, but the enclave can continue to run.
- z/OS UNIX consideration—The TERMTHDACT option applies when a thread terminates abnormally. Abnormal termination of a single thread causes termination of the entire enclave. If an unhandled condition of severity 2 or higher percolates beyond the first routine’s stack frame, the enclave terminates abnormally.

If an enclave terminates due to a POSIX default signal action, TERMTHDACT applies only to conditions that result from program checks or abends.

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

- The environment variable _CEE_DMPTARG allows a sysout class for a dynamically allocated CEEDUMP. You can set the _CEE_DMPTARG value string from a z/OS UNIX shell by:
  - using the export command
  - using the C functions setenv() or putenv()
  - using the ENVAR run-time option

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

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

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

For more information

- See "TRACE" on page 91, for more information about the TRACE run-time option.
- For more information about the CEE3DMP service and its parameters, see "CEE3DMP—Generate dump" on page 142.
- See z/OS Language Environment Programming Guide for more information about the TERMTHDACT run-time option and condition message.
- For more information about CESE, see z/OS Language Environment Programming Guide.

TEST | NOTEST

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>NOTEST(ALL,*,PROMPT,INSPPREF)</td>
</tr>
<tr>
<td>CICS</td>
<td>NOTEST(ALL,*,PROMPT,INSPPREF)</td>
</tr>
<tr>
<td>AMODE 64</td>
<td>NOTEST(ALL,*,PROMPT,INSPPREF)</td>
</tr>
</tbody>
</table>

**Syntax**

```
TEST NOTest (ALL,ERROR,commands_file,PROMPT,NOPROMPT,command,preference_file)
```

- **ALL** Specifies that any of the following causes the debug tool to gain control even without a defined AT OCCURRENCE for a particular condition or AT TERMINATION:
  - The ATTENTION function
  - Any Language Environment condition of severity 1 or above
  - Application termination

- **ERROR** Specifies that only one of the following causes the debug tool to gain control without a defined AT OCCURRENCE for a particular condition or AT TERMINATION:
  - The ATTENTION function
  - Any Language Environment-defined error condition of severity 2 or higher
Application termination

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

commands_file
A valid ddname, data set name (MVS), or file name (CMS), specifying the primary commands file for this run. If you do not specify this parameter all requests for commands go to the user terminal.

You can enclose commands_file in single or double quotes to distinguish it from the rest of the TEST | NOTEST suboption list. It can have a maximum length of 80 characters. If the data set name provided could be interpreted as a ddname, it must be preceded by a slash (/). The slash and data set name must be enclosed in quotes.

A primary commands file is required when running in a batch environment.

* (asterisk—in place of commands_file)
Specifies that no commands_file is supplied. The terminal, if available, is used as the source of the debug tool commands.

PROMPT
Specifies that the debug tool is invoked at Language Environment initialization.

NOPROMPT
Specifies that the debug tool is not invoked at Language Environment initialization.

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

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

command
A character string that specifies a valid debug tool command. The command list can be enclosed in single or double 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. The list can have a maximum of 250 characters.

INSPPREF
Specifies the default setting for preference_file.

preference_file
A valid ddname, data set name (MVS), or file name (CMS), specifying the preference file to be used. A preference file is a type of commands file that you can use to specify settings for your debugging environment. It is analogous to creating a profile for a text editor, or initializing a terminal session.

You can enclose preference_file in single or double quotes to distinguish it from the rest of the TEST parameter list. It can have a maximum of 80 characters.
TEST | NOTEST

If a specified data set name could be interpreted as a *ddname*, it must be preceded by a slash (/). The slash and data set name must be enclosed in quotes.

* (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 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.
- z/OS UNIX consideration—Language Environment honors the initial command string before the main routine runs on the initial thread.
  The test level (ALL, ERROR, NONE) applies to the enclave.
  Language Environment honors the preference file when the debug tool is initialized, regardless of which thread first requests the debug tool services.

Performance consideration

To improve performance, use this option only while debugging.

For more information

See Debug Tool publications for details and examples of the TEST run-time option as it relates to Debug Tool.

THREADHEAP

Derivation

<table>
<thead>
<tr>
<th>THREAD level HEAP storage</th>
</tr>
</thead>
</table>

THREADHEAP controls the allocation and management of thread-level heap storage. Separate heap segments are allocated and freed for each thread based on the THREADHEAP specification.

For PL/I MTF applications, controlled and based variables declared in a subtask are allocated from heap storage specified by THREADHEAP. Variables in the main task are allocated from heap storage specified by HEAP.

Library use of heap storage in a substack is allocated from the enclave-level heap storage specified by the ANYHEAP and BELOWHEAP options.

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>THREADHEAP(4K,4K,ANY,KEEP)</td>
</tr>
<tr>
<td>CICS</td>
<td>THREADHEAP is ignored under CICS.</td>
</tr>
</tbody>
</table>
Syntax

```c
THREADHeap([init_size], [incr_size], ANYWHERE, FREE, KEEP)
```

`init_size`

The minimum initial size of thread heap storage, and is specified in \( n \), \( nK \), or \( nM \). Storage is acquired in multiples of 8 bytes. A value of zero (0) causes an allocation of 4K.

`incr_size`

The minimum size of any subsequent increment to the noninitial heap storage is specified in \( n \), \( nK \), or \( nM \). The actual amount of allocated storage is the larger of two values, \( incr_size \) or the requested size, rounded up to the nearest multiple of 8 bytes. If you specify \( incr_size \) as 0, only the amount of the storage needed at the time of the request (rounded up to the nearest 8 bytes) is obtained.

`ANYWHERE | ANY`

Specifies that the heap storage can be allocated anywhere in storage. If there is no available storage above the line, storage is acquired below the 16 MB line. The only valid abbreviation of `ANYWHERE` is `ANY`.

`BELOW`

Specifies that the heap storage must be allocated below the 16M line.

`KEEP`

Specifies that storage allocated to `THREADHEAP` increments is not released when the last of the storage in the thread heap increment is freed.

`FREE`

Specifies that storage allocated to `THREADHEAP` increments is released when the last of the storage in the thread heap increment is freed.

**CICS consideration**

Even though this option is ignored under CICS, the default increment size under CICS has changed from 4K (4096 bytes) to 4080 bytes, to accommodate the 16 bytes CICS storage check zone.

**Usage notes**

- If the requesting routine is running in 24-bit addressing mode and `THREADHEAP(_,ANY)` is in effect, `THREADHEAP` storage is allocated below the 16M line based upon the `HEAP(_,initsz24,incrsz24)` settings.
- PL/1 MTF considerations—The thread-level heap is allocated only in applications that use the PL/1 MTF. For PL/1 MTF applications, controlled and based variables specified in subtasks are located in the thread-level heap.

If the main program is running in 24-bit addressing mode and `THREADHEAP(_,ANY)` is in effect, heap storage is allocated below the 16M line. The only case in which storage is allocated above the line is when all of the following conditions exist:
- The user routine requesting the storage is running in 31-bit addressing mode.
THREADHEAP

- HEAP(_,ANY) is in effect.
- The main routine is running in 31-bit addressing mode.
- When running PL/I with POSIX(ON) in effect, THREADHEAP is used for allocating heap storage for PL/I base variables declared in non-IPTs. Storage allocated to all THREADHEAP segments is freed when the thread terminates.
- THREADHEAP(4K,4K,ANYWHERE,KEEP) provides behavior compatible with the PL/I TASKHEAP option.
- The initial thread heap segment is never released until the thread terminates.
- THREADHEAP has no effect on C/C++ or Fortran for z/OS MTF applications.

THREADSTACK

**Derivation**

<table>
<thead>
<tr>
<th>THREAD level STACK storage</th>
</tr>
</thead>
</table>

THREADSTACK controls the allocation of the thread's stack storage for both the upward and downward-growing stacks, except the initial thread in a multithreaded application. If the thread attribute object does not provide an explicit stack size, then the allocation values can be inherited from the STACK option or specified explicitly on the THREADSTACK option.

**Default Value**

<table>
<thead>
<tr>
<th>Non-CICS</th>
<th>THREADSTACK(OFF,4K,4K,ANYWHERE,KEEP,128K,128K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>THREADSTACK is ignored under CICS.</td>
</tr>
</tbody>
</table>

**Syntax**

```
THREADStack(OFF, ON, usinit_size, usincr_size, ANYWHERE, ANY, BELOW, KEEP, FREE, dsinit_size, dsincr_size)
```

**OFF**

Indicates that the allocation suboptions of the STACK run-time option are used for thread stack allocation. Any other suboption specified with THREADSTACK is ignored.

**ON**

Controls the stack allocation for each thread, except the initial thread, in a multithreaded environment.

**usinit_size**

Determines the size of the initial upward-growing stack segment. The storage is contiguous. You specify the *usinit_size* value as *n*, *n*K, or *n*M bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

The *usinit_size* value can be preceded by a minus sign. In environments other than CICS, if you specify a negative number Language Environment uses all available storage minus the amount specified for the initial stack segment.
A size of "0" or "−0" requests half of the largest block of contiguous storage in the region below the 16 MB line.

`usincr_size`  
Determines the minimum size of any subsequent increment to the upward-growing stack area. You can specify this value as \( n \), \( nK \), or \( nM \) bytes of storage. The actual amount of allocated storage is the larger of two values— `usincr_size` or the requested size—rounded up to the nearest multiple of 8 bytes.

If you specify `usincr_size` as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 8 bytes, is obtained.

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

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

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

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

**FREE**  
Specifies that storage allocated to stack increments is released when the last of the storage in the stack is freed. The initial stack segment is never released until the enclave terminates.

`dsinit_size`  
Determines the size of the initial downward-growing stack segment. The storage is contiguous. You specify the `init_size` value as \( n \), \( nK \), or \( nM \) bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 16 bytes.

`dsincr_size`  
Determines the minimum size of any subsequent increment to the downward-growing stack area. You can specify this value as \( n \), \( nK \), or \( nM \) bytes of storage. The actual amount of allocated storage is the larger of two values— `incr_size` or the requested size—rounded up to the nearest multiple of 16 bytes.

**Usage notes**

- The `dsinit_size` and `dsincr_size` values are the amounts of storage that can be used for downward-growing stack frames (plus the stack header, approximately 20 bytes). The actual size of the storage getmained will be 4K (8K if a 4K page alignment cannot be guaranteed) larger to accommodate the guard page.
**THREADSTACK**

- The downward-growing stack is only initialized in a XPLINK supported environment, that is, batch, TSO, z/OS UNIX, and only when a XPLINK application is active in the enclave. Otherwise the suboptions for the downward-growing stack are ignored.
- All storage allocated to THREADSTACK segments are freed when the thread terminates.
- The initial stack segment of the thread is never released until the thread terminates, regardless of the KEEP/FREE state.
- You can specify sub-options with THREADSTACK(OFF,...), but they are ignored. If you override the THREADSTACK(OFF,...) suboption with THREADSTACK(ON) and you omit suboptions, then the suboptions you specified with THREADSTACK(OFF,...) remain in effect. If you respecify THREADSTACK(OFF,...) with different suboptions, they override the defaults.
- PL/I MTF consideration—THREADSTACK(ON,4K,4K,BELOW,KEEP,) provides PL/I compatibility for stack storage allocation and management for each subtask in the application.
- PL/I considerations—For multitasking or multithreaded environments, the stack size for a subtask or non-Initial Process Thread (non-IPT) is taken from the THREADSTACK option unless THREADSTACK(OFF) is specified. THREADSTACK(OFF) specifies that the values in the STACK option be used.
- In the multithreaded environment, you can explicitly specify the stack size in the thread attribute object; it will be used instead of the value specified with THREADSTACK or STACK.
- The THREADSTACK option replaces the NONIPTSTACK and NONNONIPTSTACK options.

**THREADSTACK64 (AMODE 64 only)**

<table>
<thead>
<tr>
<th>Derivation</th>
<th>THREAD level STACK storage</th>
</tr>
</thead>
</table>

THREADSTACK64 controls the allocation of the thread's stack storage for AMODE 64 applications, except for the initial thread in a multithreaded environment.

**Default Value**

| AMODE 64 | THREADSTACK64(OFF,1M,1M,128M) |

**Syntax**

```
THREADSTACK64(OF,ON,initial,increment,maximum)
```

**OFF**

Indicates that the allocation suboptions of the STACK64 run-time option are used for thread stack allocation. Any other suboption specified with THREADSTACK64 is ignored.

**ON**

Controls the stack allocation for each thread, except the initial thread, in a multithreaded environment.
**THREADSTACK64**

initial

Determines the size of the initial stack segment. The storage is contiguous. This value is specified as $nM$ bytes of storage.

increment

Determines the minimum size of any subsequent increment to the stack area. This value is specified as $nM$ bytes of storage. The actual amount of allocated storage is the larger of two values—**increment** or the requested size—rounded up to the nearest multiple of 1MB.

If you specify **increment** as 0, only the amount of the storage needed at the time of the request, rounded up to the nearest multiple of 1MB, is obtained.

The requested size is the amount of storage a routine needs for a stack frame.

maximum

Specifies the maximum stack size. This value is specified as $nM$ bytes of storage. When the maximum size is less than the initial size, **initial** is used as the maximum stack size.

**Usage notes**

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

For more information

For more information about heap storage and heap storage tuning with storage report numbers, see [z/OS Language Environment Programming Guide](https://www.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.zos.v1r11.sed.rte.doc/rte_ch02.html)?

**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 Language Environment trace entries in the trace table.

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>TRACE(OFF,4K,DUMP;LE=0)</td>
</tr>
<tr>
<td>CICS</td>
<td>TRACE(OFF,4K,DUMP;LE=0)</td>
</tr>
</tbody>
</table>
TRACE

AMODE 64  TRACE(OFF,DUMP,LE=0)

Syntax

```
/SM590000/SM590000
TRACe (OFF, ON, table_size, DUMP, NODUMP, LE=0, LE=1, LE=2, LE=3, LE=20)
```

<table>
<thead>
<tr>
<th>OFF</th>
<th>Indicates that the tracing facility is inactive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Indicates that the tracing facility is active.</td>
</tr>
<tr>
<td><code>table_size</code></td>
<td>Determines the size of the tracing table as specified in bytes (nK or nM). The upper limit is 16M - 1 (1666777215 bytes). <strong>Restriction:</strong> This option is ignored in a 64-bit environment and the size is set to 1M.</td>
</tr>
<tr>
<td>DUMP</td>
<td>Requests that a Language Environment-formatted dump (containing the trace table) be taken at program termination regardless of the setting of the TERMTHDACT run-time option.</td>
</tr>
<tr>
<td>NODUMP</td>
<td>Requests that a Language Environment-formatted dump not be taken at program termination.</td>
</tr>
<tr>
<td>LE=0</td>
<td>Specifies that no trace events be recorded.</td>
</tr>
<tr>
<td>LE=1</td>
<td>Specifies that entry to and exit from Language Environment member libraries be recorded (such as, in the case of C, entry and exit of the <code>printf()</code> library function).</td>
</tr>
<tr>
<td>LE=2</td>
<td>Specifies that mutex init/destroy and locks/unlocks from Language Environment member libraries be recorded.</td>
</tr>
<tr>
<td>LE=3</td>
<td>Activates both the entry/exit trace and the mutex trace.</td>
</tr>
<tr>
<td>LE=20</td>
<td>Specifies that XPLINK/non-XPLINK transition should be recorded.</td>
</tr>
</tbody>
</table>

**Usage notes**

- **PL/I MTF consideration**—The TRACE(ON,,LE=2) setting provides the following trace table entries for PL/I MTF support:
  - Trace entry 100 occurs when a task is created.
  - Trace entry 101 occurs when a task that contains the tasking CALL statements is terminated.
  - Trace entry 102 occurs when a task that does not contain the tasking CALL statements is terminated.
- When running PL/I with POSIX(ON) in effect, no PL/I-specific trace information is provided.
- **When you specify LE=20:**
  - AMODE 64 applications have no transitions.
  - Transitions across OS_UPSTACK linkage are not recorded.
- **When TRACE(OFF) is specified,** Language Environment activates tracing when a `pthread_create()` is done and a debugger is being used.
- COBOL does not provide any Trace Table Entries.
TRAP

TRAP specifies how Language Environment programs handle abends and program interrupts (see Table 7). CEESGL is unaffected by this option.

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

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

Table 7. TRAP Run-Time Option Settings

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

Default Value

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
</tr>
<tr>
<td>CICS</td>
</tr>
<tr>
<td>AMODE 64</td>
</tr>
</tbody>
</table>

Syntax

```plaintext
/SM590000/SM630000
TRAP (ON, OFF, SPIE, NOSPIE)
```

ON Fully enables the Language Environment condition handler.

OFF Prevents language condition handlers or handlers registered by CEEHDLR from being notified of abends or program checks; prevents application of POSIX signal handling semantics for abends and program checks.
TRAP

SPIE  SPIE specifies that Language Environment issue an ESPIE macro to handle program interrupts. The SPIE sub-option is ignored when specified with the OFF sub-option.

NOSPIE  NOSPIE specifies that Language Environment will NOT issue the ESPIE macro. When you specify the ON, NOSPIE sub-option, Language Environment will handle program interrupts and abends via an ESTAE. The NOSPIE sub-option is ignored when specified with the OFF sub-option.

Due to the restrictions and side-effects when running TRAP(OFF) stated in the "Usage notes," IBM highly recommends running TRAP(ON,SPIE) in all environments.

CICS consideration
Since Language Environment never sets a SPIE or STAE, the SPIE|NOSPIE sub-option is ignored on CICS.

Usage notes
• Use TRAP(OFF) only when you need to analyze a program exception before Language Environment handles it.
• When you specify TRAP(OFF) in a non-CICS environment, an ESPIE is not issued, but an ESTAE is issued. Language Environment does not handle conditions raised by program interrupts or abends initiated by SVC 13 as Language Environment conditions, and does not print messages for such conditions.
• Language Environment requires TRAP(ON) because it uses condition handling internally. As a result, running with TRAP(OFF) for exception diagnosis purposes can cause many side effects. If you run with TRAP(OFF), you can get side effects even if you do not encounter a software-raised condition, program check, or abend. If you do encounter a program check or an abend with TRAP(OFF) in effect, the following side effects can occur:
  – The ABTERMENC run-time option has no effect.
  – The ABPERC run-time option has no effect.
  – Resources acquired by Language Environment are not freed.
  – Files opened by HLLs are not closed by Language Environment, so records might be lost.
  – The abnormal termination exit is not driven for enclave termination.
  – The assembler user exit is not driven for enclave termination.
  – User condition handlers are not enabled.
  – The debugger is not notified of the error.
  – No storage report or run-time options report is generated.
  – No Language Environment messages or Language Environment dump output is generated.
  – In z/OS UNIX, POSIX signal handling semantics are not enabled for the abend.
  – The enclave terminates abnormally if such conditions are raised.
  – The COBOL ON SIZE ERROR clause may be disabled for arithmetic statements.
TRAP

- TRAP(ON) must be in effect when you use the CEEBXITA assembler user exit for enclave initialization to specify a list of abend codes that Language Environment percolates.
- C++ consideration—TRAP(ON) must be in effect in order for the z/OS C++ try/throw/catch condition handling mechanisms to work.
- When TRAP(ON) is in effect, and the abend code is in the CEEAUE_CODEES list in CEEBXITA, Language Environment percolates the abend. Normal Language Environment condition handling is never invoked to handle these abends. This feature is useful when you do not want Language Environment condition handling to intervene for certain abends or when you want to prevent invocation of the abnormal termination exit for certain abends, such as when IMS issues a user ABEND code 777.
- When TRAP(ON,NOSPIE) is specified in a non-CICS environment, Language Environment will handle program interrupts and abends via an ESTAE. This feature is useful when you do not want Language Environment to issue an ESPIE macro.
- When TRAP(OFF), (TRAP(OFF,SPIE) or TRAP(OFF,NOSPIE) is specified and there is a program interrupt, the user exit for termination is not driven.
- z/OS UNIX consideration—The TRAP option applies to the entire enclave and all threads within.

For more information
- See "ABTERMENC" on page 10 for more information about the ABTERMENC run-time option.
- See "CEESGL—Signal a condition" on page 453 for more information about the CEESGL callable service.
- For more information about the CEEHDLR callable service, see "CEEHDLR—Register User-written condition handler" on page 336.
- See z/OS Language Environment Programming Guide for more information about the CEEBXITA assembler user exit.

UPSI (COBOL only)

<table>
<thead>
<tr>
<th>Derivation</th>
<th>User Programmable Status Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPSI</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>UPSI(00000000)</td>
</tr>
<tr>
<td>CICS</td>
<td>UPSI(00000000)</td>
</tr>
</tbody>
</table>

Syntax

```
UPSII(00000000)
```
**UPSI**

\[ \text{nnnnnnnn} \]

\( n \) 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).

**CICS consideration**

The default under CICS is UPSI=(00000000).

**Usage note**

Specify this option in CEEDOPT(CEECOPT), CEEROPT or CEEUOPT with a string of eight binary 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 programs access the UPSI switches, see [Enterprise COBOL for z/OS, V4R2, Programming Guide](SC23-8529), [Enterprise COBOL for z/OS, V3R4, Programming Guide](SC27-1412), or [COBOL for OS/390 & VM Programming Guide](SC23-8529).

---

**USRHDLR | NOUSRHDLR**

**Derivation**

User condition handler

USRHDLR registers a user condition handler at stack frame 0, allowing you to register a user condition handler without having to include a call to CEEHDLR in your application and then recompile the application.

**Default Value**

- **Non-CICS**: NOUSRHDLR
- **CICS**: NOUSRHDLR()

**Syntax**

```
USRHDLR (lmname, lmname2)
```

**NOUSRHDLR**

Does not register a user condition handler without recompiling an application to include a call to CEEHDLR.

**USRHDLR**

Registers a user condition handler without recompiling an application to include a call to CEEHDLR.

**lmname**

The name of a load module (or an alias name of a load module) that contains the user condition handler that is to be registered at stack frame 0. This parameter is optional.

**lmname2**

The name of a load module (or an alias name of a load module) that
USRHDLR | NOUSRHDLR

contains the user condition handler that is to be registered to get control after the enablement phase and before any other user condition handler. This parameter is optional.

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

Usage notes
- The user condition handler specified by the USRHDLR run-time option must be in a separate load module rather than be link-edited with the rest of the application.
- The user condition handler \texttt{lmname} is invoked for conditions that are still unhandled after being presented to condition handlers for the main program.
- The user condition handler \texttt{lmname2} is invoked for each condition after the condition completes the enablement phase but before any other registered user condition handlers are given control.
- You can use a user condition handler registered with the USRHDLR 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 to handle conditions at stack frame 0 using the USRHDLR run time option is not presented to any other user condition handler.
- The loading of the user condition handlers \texttt{lmname} and \texttt{lmname2} occurs only when that user condition handler needs to be invoked the first time.
- If the load of either \texttt{lmname} or \texttt{lmname2} fails, an error message is issued.
- To turn off one of the sub-options previously specified by USRHDLR (\texttt{lmname} or \texttt{lmname2}), specify the option with either empty single quotes or empty double quotes. For example, to turn off the \texttt{lmname2} sub-option after it had been previously specified, use either USRHDLR\texttt{(lmname,‘’)} or USRHDLR\texttt{(lmname,‘”’)}.
- In SCEESAMP, IBM supplies a sample user-written condition handler, called CEEWUCHA. Under CICS, this handler will give you similar abend codes that existed in certain pre-Language Environment environments. The CEEWUCHA load module needs to be built using CEEWWCHA, which is also provided in SCEESAMP. Note that this handler has support for both COBOL and PL/I; however, it is shipped with the PL/I-specific behavior commented out. If you want this PL/I behavior, modify the source before using CEEWWCHA.

For more information
For information on registering a user condition handler and its interfaces, see "CEEHDLR—Register User-written condition handler" on page 336.

VCTRSAVE

<table>
<thead>
<tr>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VecToR environment to be SAVEd</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>VCTRSAVE(OFF)</td>
</tr>
<tr>
<td>CICS</td>
<td>VCTRSAVE is ignored under CICS.</td>
</tr>
</tbody>
</table>

**Syntax**

```
Vctrsave (OFF)
```

**OFF**  
No language in the application uses the vector facility when user-provided condition handlers are called.

**ON**  
A language in the application uses the vector facility when user-provided condition handlers are called.

**CICS consideration**

VCTRSAVE is ignored under CICS.

**z/OS UNIX consideration**

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

**Performance consideration**

When a condition handler plans to use the vector facility (that is, run any vector instructions), the entire vector environment has to be saved on every condition and restored on return to the application code. You can avoid this extra work by specifying VCTRSAVE(OFF) when you are not running an application under vector hardware.

**XPLINK**

<table>
<thead>
<tr>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXtra Performance LINKage</td>
</tr>
</tbody>
</table>

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

You cannot set XPLINK during Language Environment installation (CEEDOPT/CEEROPT) or from the CEEBXITA Assembler User Exit Interface. You
can only specify XPLINK on application invocation (command line interface or _CEE_RUNOPTS environment variable), or as an application default (CEEUOPT, C/C++ #pragma runopts, or PLIXOPT).

**Default**

**Non-CICS**
XPLINK(OFF)

**CICS**
XPLINK is ignored under CICS.

**Syntax**

```
XPLink ((OFF | ON))
```

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

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

**Usage notes**

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

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

For these reasons, the XPLINK(ON) application should only be specified for individual applications that require it. That is why you cannot set XPLINK(ON) as the installation default (eg. in CEEDOPT), and why you should not export _CEE_RUNOPTS from the UNIX System Services shell containing the XPLINK(ON) run-time option.

- If the XPLINK run-time option is not specified and the initial program contains at least one XPLINK-compiled part, then the XPLINK run-time option will be forced to (ON). No message will be issued to indicate this action.
- When an application is running in an XPLINK environment (that is, either the XPLINK(ON) run-time option was specified, or the initial program contained at least one XPLINK compiled part), the ALL31 run-time option will be forced to ON. No AMODE 24 routines are allowed in an enclave that uses XPLINK. No message will be issued to indicate this action. If a Language Environment run-time options report is generated using the RPTOPTS run-time option, the ALL31 option will be reported as "Override" under the LAST WHERE SET column.
- When an application is running in an XPLINK environment (that is, either the XPLINK(ON) run-time option was specified, or the initial program contained at least one XPLINK compiled part), the STACK run-time option will be forced to STACK(,,ANY). Only the third suboption is changed by this action to indicate
that STACK storage can be allocated anywhere in storage. No message will be
issued to indicate this action. If a Language Environment run-time options
report is generated using the RPTOPTS run-time option, the STACK option will
be reported as "Override" under the LAST WHERE SET column.

Performance Considerations

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

XUFLOW

<table>
<thead>
<tr>
<th>Derivation</th>
<th>XUFLOW(AUTO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXponent Under FLOW</td>
<td></td>
</tr>
</tbody>
</table>

XUFLOW specifies if an exponent underflow causes a program interrupt. An
exponent underflow occurs when a floating point number becomes too small to be
represented. The underflow setting is determined at enclave initialization and is
updated when new languages are introduced into the application (via fetch or
dynamic call, for example). Otherwise, it does not vary while the application is
running.

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

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-CICS</td>
<td>XUFLOW(AUTO)</td>
</tr>
<tr>
<td>CICS</td>
<td>XUFLOW(AUTO)</td>
</tr>
</tbody>
</table>
Syntax

```
XUFLOW(ON)
```

**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.
- z/OS UNIX consideration—The XUFLOW option applies to the entire enclave and all threads within.

**Language run-time option mapping**

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

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

Tables of pre-Language Environment, HLL-specific options and their Language Environment equivalents are provided in the [z/OS Language Environment Run-Time Application Migration Guide](#). The mapping is performed automatically by Language Environment, except where noted.
Part 2. Language Environment callable services

The following sections contain detailed information about how to use the Language Environment callable services.
Chapter 3. Quick reference tables for Language Environment services

The following tables are a quick reference of the Language Environment callable services and math services. They list the location of the services and briefly state their function. For detailed descriptions of these services, refer to their respective sections in this book.

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

**Bit manipulation routines**

*Table 8. Bit Manipulation Routines*

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESICLR</td>
<td>Bit clear</td>
<td>471</td>
</tr>
<tr>
<td>CEESISET</td>
<td>Bit set</td>
<td>471</td>
</tr>
<tr>
<td>CEESISHF</td>
<td>Bit shift</td>
<td>472</td>
</tr>
<tr>
<td>CEESITST</td>
<td>Bit test</td>
<td>473</td>
</tr>
</tbody>
</table>

**Condition handling callable services**

*Table 9. Condition Handling Callable Services*

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE3CIB</td>
<td>Returns a pointer to a condition information block (CIB) associated with a given condition token.</td>
<td>128</td>
</tr>
<tr>
<td>CEE3GRN</td>
<td>Gets the name of the most current Language Environment-conforming routine in which a condition occurred.</td>
<td>158</td>
</tr>
<tr>
<td>CEE3GRO</td>
<td>Returns the offset within a failing routine of the most recent condition.</td>
<td>164</td>
</tr>
<tr>
<td>CEE3SPM</td>
<td>Queries and modifies the enablement of Language Environment hardware conditions.</td>
<td>206</td>
</tr>
<tr>
<td>CEE3SRP</td>
<td>Sets the resume point at the next instruction in the calling routine to be used by CEEMRCE.</td>
<td>213</td>
</tr>
<tr>
<td>CEEDCOD</td>
<td>Decodes an existing condition token.</td>
<td>256</td>
</tr>
<tr>
<td>CEEENV</td>
<td>Processes environment variables based on the function codes passed as input.</td>
<td>277</td>
</tr>
<tr>
<td>CEEGQDT</td>
<td>Provides a mechanism by which application code can retrieve the $q_{\text{data}}$ token from the Instance Specific Information (ISI).</td>
<td>321</td>
</tr>
<tr>
<td>CEEHDLR</td>
<td>Registers a user condition handler for the current stack frame.</td>
<td>336</td>
</tr>
<tr>
<td>CEEHDLU</td>
<td>Unregisters a user-written condition handler for the current stack frame.</td>
<td>346</td>
</tr>
<tr>
<td>CEEITOK</td>
<td>Computes the initial condition token for the current condition information block.</td>
<td>357</td>
</tr>
<tr>
<td>CEEMRCE</td>
<td>Resumes execution of a user routine at the location established by CEE3SRP.</td>
<td>384</td>
</tr>
<tr>
<td>CEEMRCR</td>
<td>Moves the resume cursor to a position relative to the current position of the handle cursor.</td>
<td>391</td>
</tr>
<tr>
<td>CEENCOD</td>
<td>Dynamically constructs a condition token.</td>
<td>407</td>
</tr>
</tbody>
</table>
Table 9. Condition Handling Callable Services (continued)

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESGL</td>
<td>Raises, or signals, a condition to the condition manager.</td>
<td>453</td>
</tr>
</tbody>
</table>

**Date and time callable services**

Table 10. Date and Time Callable Services

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEECBLDY</td>
<td>Converts a string representing a date to a COBOL integer format.</td>
<td>218</td>
</tr>
<tr>
<td>CEEDATE</td>
<td>Converts a number representing a Lilian date to a date written in character format.</td>
<td>237</td>
</tr>
<tr>
<td>CEEDATM</td>
<td>Converts a number representing the number of seconds since 00:00:00 14 October 1582 to character format.</td>
<td>243</td>
</tr>
<tr>
<td>CEE DAYS</td>
<td>Converts a string representing a date to a Lilian format.</td>
<td>250</td>
</tr>
<tr>
<td>CEE DYWK</td>
<td>Calculates the day of the week on which a Lilian date falls.</td>
<td>272</td>
</tr>
<tr>
<td>CEE GMT</td>
<td>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.</td>
<td>309</td>
</tr>
<tr>
<td>CEE GMTO</td>
<td>Computes values to the calling routine that represent the difference between the local system time and Greenwich Mean Time.</td>
<td>313</td>
</tr>
<tr>
<td>CEE IS EC</td>
<td>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.</td>
<td>351</td>
</tr>
<tr>
<td>CEE LO CT</td>
<td>Gets the current local time in three formats.</td>
<td>370</td>
</tr>
<tr>
<td>CEE QC EN</td>
<td>Queries the century within which Language Environment assumes two-digit year values lie.</td>
<td>414</td>
</tr>
<tr>
<td>CEE SC EN</td>
<td>Sets the century in which Language Environment assumes two-digit year values lie.</td>
<td>428</td>
</tr>
<tr>
<td>CEE SE CI</td>
<td>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.</td>
<td>436</td>
</tr>
<tr>
<td>CEE SE CS</td>
<td>Converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582.</td>
<td>441</td>
</tr>
</tbody>
</table>

**Dynamic storage callable services**

Table 11. Dynamic Storage Callable Services

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE 3 RPH</td>
<td>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.</td>
<td>203</td>
</tr>
<tr>
<td>CEE CR HP</td>
<td>Creates additional heaps.</td>
<td>227</td>
</tr>
<tr>
<td>CEE CZ ST</td>
<td>Changes the size of a previously allocated storage element, while preserving its contents.</td>
<td>232</td>
</tr>
<tr>
<td>CEE DS HP</td>
<td>Discards an entire heap that you created previously with a call to CEE CR HP.</td>
<td>268</td>
</tr>
<tr>
<td>CEE FR ST</td>
<td>Frees storage previously allocated by CEE GT ST.</td>
<td>298</td>
</tr>
<tr>
<td>CEE GT ST</td>
<td>Allocates storage from a heap whose ID you specify.</td>
<td>331</td>
</tr>
</tbody>
</table>
# General callable services

**Table 12. General Callable Services**

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE3DLY</td>
<td>Suspends processing of an active enclave for a specified number of seconds.</td>
<td>138</td>
</tr>
<tr>
<td>CEE3DMP</td>
<td>Generates a Language Environment formatted dump of the run-time environment and member language libraries.</td>
<td>142</td>
</tr>
<tr>
<td>CEE3INF</td>
<td>Returns to the calling routine information about the system, susbsystem, environment, member languages, and version of Language Environment associated with the enclave.</td>
<td>169</td>
</tr>
<tr>
<td>CEE3PRM</td>
<td>Returns to the calling routine the argument string that was specified at program invocation.</td>
<td>195</td>
</tr>
<tr>
<td>CEE3PR2</td>
<td>Returns to the calling routine the argument string and its associated length that was specified at program invocation.</td>
<td>195</td>
</tr>
<tr>
<td>CEE3USR</td>
<td>Sets or queries one of two 4-byte fields known as the user area fields.</td>
<td>214</td>
</tr>
<tr>
<td>CEEEDLYM</td>
<td>Suspends processing of an active enclave for a specified number of milliseconds.</td>
<td>262</td>
</tr>
<tr>
<td>CEEGPID</td>
<td>Retrieves the version of Language Environment currently in use.</td>
<td>316</td>
</tr>
<tr>
<td>CEERAN0</td>
<td>Generates a sequence of uniform pseudo-random numbers between 0.0 and 1.0 using the multiplicative congruential method with a user-specified seed.</td>
<td>425</td>
</tr>
<tr>
<td>CEETEST</td>
<td>Invokes a tool for debugging.</td>
<td>466</td>
</tr>
</tbody>
</table>

# Initialization and termination services

**Table 13. Initialization and Termination Services**

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE3ABD</td>
<td>Stops the enclave with an abend. The abend can be issued either with or without clean-up.</td>
<td>121</td>
</tr>
<tr>
<td>CEE3AB2</td>
<td>Stops the enclave with an abend and a reason code. The abend can be issued either with or without clean-up.</td>
<td>124</td>
</tr>
<tr>
<td>CEE3GRC</td>
<td>Gets the enclave return code.</td>
<td>148</td>
</tr>
<tr>
<td>CEE3SRC</td>
<td>Sets the enclave return code.</td>
<td>212</td>
</tr>
</tbody>
</table>

# Locale callable services

**Table 14. Locale Callable Services**

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEEFMON</td>
<td>Formats monetary strings.</td>
<td>290</td>
</tr>
<tr>
<td>CEEFTDS</td>
<td>Formats time and date into a character string.</td>
<td>303</td>
</tr>
<tr>
<td>CEELCNV</td>
<td>Queries locale numeric conventions.</td>
<td>363</td>
</tr>
<tr>
<td>CEEQDTC</td>
<td>Queries locale date and time conventions and returns the specified format information.</td>
<td>417</td>
</tr>
<tr>
<td>CEEQRYL</td>
<td>Queries the active locale environment.</td>
<td>423</td>
</tr>
<tr>
<td>CEESCOL</td>
<td>Compares the collation weights of two strings.</td>
<td>432</td>
</tr>
<tr>
<td>CEESETL</td>
<td>Sets the locale operating environment.</td>
<td>448</td>
</tr>
<tr>
<td>CEESTXF</td>
<td>Transforms string characters into collation weights.</td>
<td>459</td>
</tr>
</tbody>
</table>
## Math services

### Table 15. Math Services

<table>
<thead>
<tr>
<th>Math Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEEsxABS</td>
<td>Absolute value</td>
<td>477</td>
</tr>
<tr>
<td>CEEsxACS</td>
<td>Arccosine</td>
<td>478</td>
</tr>
<tr>
<td>CEEsxASN</td>
<td>Arcsine</td>
<td>479</td>
</tr>
<tr>
<td>CEEsxATH</td>
<td>Hyperbolic arctangent</td>
<td>480</td>
</tr>
<tr>
<td>CEEsxATN</td>
<td>Arctangent</td>
<td>481</td>
</tr>
<tr>
<td>CEEsxAT2</td>
<td>Arctangent of two arguments</td>
<td>482</td>
</tr>
<tr>
<td>CEEsxCJG</td>
<td>Conjugate complex</td>
<td>483</td>
</tr>
<tr>
<td>CEEsxCOS</td>
<td>Cosine</td>
<td>484</td>
</tr>
<tr>
<td>CEEsxCSH</td>
<td>Hyperbolic cosine</td>
<td>485</td>
</tr>
<tr>
<td>CEEsxCTN</td>
<td>Cotangent</td>
<td>486</td>
</tr>
<tr>
<td>CEEsxDIM</td>
<td>Positive difference</td>
<td>487</td>
</tr>
<tr>
<td>CEEsxDVD</td>
<td>Division</td>
<td>488</td>
</tr>
<tr>
<td>CEEsxERC</td>
<td>Error function complement</td>
<td>489</td>
</tr>
<tr>
<td>CEEsxERF</td>
<td>Error function</td>
<td>490</td>
</tr>
<tr>
<td>CEEsxEXP</td>
<td>Exponential (base e)</td>
<td>490</td>
</tr>
<tr>
<td>CEEsxGMA</td>
<td>Gamma function</td>
<td>492</td>
</tr>
<tr>
<td>CEEsxIMG</td>
<td>Imaginary part of complex</td>
<td>493</td>
</tr>
<tr>
<td>CEEsxINT</td>
<td>Truncation</td>
<td>493</td>
</tr>
<tr>
<td>CEEsxLGM</td>
<td>Log gamma function</td>
<td>494</td>
</tr>
<tr>
<td>CEEsxLG1</td>
<td>Logarithm base 10</td>
<td>495</td>
</tr>
<tr>
<td>CEEsxLG2</td>
<td>Logarithm base 2</td>
<td>496</td>
</tr>
<tr>
<td>CEEsxLOG</td>
<td>Logarithm base e</td>
<td>497</td>
</tr>
<tr>
<td>CEEsxMLT</td>
<td>Floating-point complex muliplication</td>
<td>498</td>
</tr>
<tr>
<td>CEEsxMOD</td>
<td>Modular arithmetic</td>
<td>498</td>
</tr>
<tr>
<td>CEEsxNIN</td>
<td>Nearest integer</td>
<td>499</td>
</tr>
<tr>
<td>CEEsxNWN</td>
<td>Nearest whole number</td>
<td>500</td>
</tr>
<tr>
<td>CEEsxSGN</td>
<td>Transfer of sign</td>
<td>501</td>
</tr>
<tr>
<td>CEEsxSIN</td>
<td>Sine</td>
<td>502</td>
</tr>
<tr>
<td>CEEsxSNH</td>
<td>Hyperbolic sine</td>
<td>503</td>
</tr>
<tr>
<td>CEEsxSQT</td>
<td>Square root</td>
<td>504</td>
</tr>
<tr>
<td>CEEsxTAN</td>
<td>Tangent</td>
<td>505</td>
</tr>
<tr>
<td>CEEsxTNH</td>
<td>Hyperbolic tangent</td>
<td>506</td>
</tr>
<tr>
<td>CEEsxXPx</td>
<td>Exponential (**)</td>
<td>507</td>
</tr>
</tbody>
</table>
### Message handling callable services

**Table 16. Message Handling Callable Services**

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEECMII</td>
<td>Stores message insert data.</td>
<td>222</td>
</tr>
<tr>
<td>CEEMGET</td>
<td>Retrieves a message corresponding to a condition token returned from a callable service.</td>
<td>374</td>
</tr>
<tr>
<td>CEEMOUT</td>
<td>Writes a specified message to the message string file.</td>
<td>380</td>
</tr>
<tr>
<td>CEEMSG</td>
<td>Retrieves a message corresponding to a condition token received and writes it to the message file.</td>
<td>402</td>
</tr>
</tbody>
</table>

### National Language Support callable services

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

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE3CTY</td>
<td>Changes or queries the current national country setting.</td>
<td>132</td>
</tr>
<tr>
<td>CEE3LNG</td>
<td>Changes or queries the current national language.</td>
<td>174</td>
</tr>
<tr>
<td>CEE3MCS</td>
<td>Gets the default currency symbol for a country specified in <code>country_code</code>.</td>
<td>181</td>
</tr>
<tr>
<td>CEE3MC2</td>
<td>Gets the default currency symbol and the international currency symbol for a country specified in <code>country_code</code>.</td>
<td>185</td>
</tr>
<tr>
<td>CEE3MDS</td>
<td>Gets the default decimal separator for the country specified in <code>country_code</code>.</td>
<td>189</td>
</tr>
<tr>
<td>CEE3MTS</td>
<td>Gets the default thousands separator for the country that you specify in <code>country_code</code>.</td>
<td>192</td>
</tr>
<tr>
<td>CEEFMDA</td>
<td>Gets the default date picture string for a country specified in <code>country_code</code>.</td>
<td>284</td>
</tr>
<tr>
<td>CEEFMDT</td>
<td>Gets the default date and time picture strings for the country specified in <code>country_code</code>.</td>
<td>287</td>
</tr>
<tr>
<td>CEEFMTM</td>
<td>Gets the default time picture string for the country specified in <code>country_code</code>.</td>
<td>295</td>
</tr>
</tbody>
</table>
Chapter 4. Language Environment callable services

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

This chapter provides syntax and examples of Language Environment callable services, which you can invoke from applications generated with the following IBM compiler products:

- IBM z/OS XL C/C++
- C/C++ for MVS/ESA
- IBM SAA AD/Cycle C/370
- Enterprise COBOL for z/OS
- Enterprise PL/I for z/OS
- IBM COBOL for OS/390 & VM
- IBM COBOL for MVS & VM
- IBM PL/I for MVS & VM

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

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

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

Locating callable service information

This chapter contains the following sections to help you use the Language Environment callable services. For a summary of Language Environment callable services, see Chapter 1, “Language Environment run-time option reference tables and general information,” on page 3.

- “General usage notes for callable services” on page 112
- “Invoking callable services” on page 112
- “Data type definitions” on page 117
- “ Callable services” on page 121
General usage notes for callable services

- You can invoke callable services from any Language Environment-conforming HLL except where otherwise noted.
- You might receive feedback codes from services other than the one you are invoking. This is because the callable services invoke other callable services that might return a feedback code.
- Callable services that are intended to be available on any platform that Language Environment supports are prefixed with CEE. Callable services defined only for S/370 are prefixed with CEE3.
- Routines that invoke callable services do not need to be in 31-bit addressing mode. 31-bit addressing mode switching is performed implicitly without any action required by the calling routine, if you specify the ALL31(OFF) run-time option (see "ALL31" on page 12). However, if AMODE switching occurs and your program makes many calls to Language Environment callable services, the switching time can slow down your application. Run with AMODE(31), if possible, to avoid unnecessary mode switching.
- Under Language Environment, all parms are passed by reference, indirectly. The code in Figure 3 and Figure 4 might cause unpredictable results:

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

**Figure 3. An Invalid COBOL CALL that Omits the fc Parameter**

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

**Figure 4. An Invalid PL/I CALL that Omits the fc Parameter**

Figure 5 and Figure 6 illustrate valid calls. The COBOL examples are valid only for COBOL for MVS & VM Release 2 or later.

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

**Figure 5. Valid COBOL CALLs that Use the Optional fc Parameter**

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

**Figure 6. Valid PL/I CALLs that Use the Optional fc Parameter**

Invoking callable services

You can invoke Language Environment callable services from assembler routines, HLL-generated object code, HLL library routines, other Language Environment library routines, and user-written HLL calls. When you want to access Language Environment library routines, you can use the same type of user-written HLL calls and functions you currently use for your HLL applications.
This section provides syntax and examples to help you request callable services from C/C++, COBOL, and PL/I.

**Header, copy, or include files**

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

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

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceeedct.h</td>
<td>C declarations for Language Environment symbolic feedback codes</td>
</tr>
<tr>
<td>leawi.h</td>
<td>Declarations of Language Environment callable services and OMIT_FC, which is</td>
</tr>
<tr>
<td></td>
<td>used to explicitly omit the fc parm, for routines written in C/C++</td>
</tr>
<tr>
<td>CEEIGZCI</td>
<td>Declarations for condition information block</td>
</tr>
<tr>
<td>CEEIGZCT</td>
<td>Declarations for Language Environment symbolic feedback codes</td>
</tr>
<tr>
<td>CEEIGZLC</td>
<td>Locale category constants LC-ALL, LC-COLLATE, LC-CTYPE, LC-MESSAGES, LC-MONETARY, LC-NUMERIC, LC-TIME, and version_info structure.</td>
</tr>
<tr>
<td>CEEIGZTD</td>
<td>TD-STRUCT structure needed for CEEFTDS calls</td>
</tr>
<tr>
<td>CEEIGZNM</td>
<td>NM-STRUCT structure needed for CEELCNV calls</td>
</tr>
<tr>
<td>CEEIGZN2</td>
<td>NM-STRUCT structure needed for ISO/IEC 9899:1999 (C99) support of CEELCNV calls</td>
</tr>
<tr>
<td>CEEIGZDT</td>
<td>DTCONV structure needed for CEEQDTC calls</td>
</tr>
<tr>
<td>CEEIGZD2</td>
<td>DTCONV structure needed for C99 support of CEEQDTC calls</td>
</tr>
<tr>
<td>CEEIBMAW</td>
<td>Language Environment callable service declarations for routines written in PL/I</td>
</tr>
<tr>
<td>CEEIBMCI</td>
<td>Declarations for condition information block</td>
</tr>
<tr>
<td>CEEIBMCT</td>
<td>Declarations for Language Environment symbolic feedback codes</td>
</tr>
<tr>
<td>CEEIBMLC</td>
<td>Locale category constants LC-ALL, LC-COLLATE, LC-CTYPE, LC-MESSAGES, LC-MONETARY, LC-NUMERIC, LC-TIME, and version_info structure.</td>
</tr>
<tr>
<td>CEEIBMTD</td>
<td>TD-STRUCT structure needed for CEEFTDS calls</td>
</tr>
<tr>
<td>CEEIBMNM</td>
<td>NM-STRUCT structure needed for CEELCNV calls</td>
</tr>
<tr>
<td>CEEIBMN2</td>
<td>NM-STRUCT structure needed for C99 support of CEELCNV calls</td>
</tr>
<tr>
<td>CEEIBMDT</td>
<td>DTCONV structure needed for CEEQDTC calls</td>
</tr>
<tr>
<td>CEEIBMD2</td>
<td>DTCONV structure needed for C99 support of CEEQDTC calls</td>
</tr>
<tr>
<td>CEEFORCT</td>
<td>Fortran declarations for Language Environment symbolic feedback codes</td>
</tr>
</tbody>
</table>
Table 18. Files Used in C/C++, COBOL, and PL/I Examples (continued)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
</table>

Notes:
1. A symbolic feedback code is a symbolic representation of a condition token.
2. PL/I routines that imbed CEEIBMAW require the MACRO compiler option.
3. COBOL does not require declarations of external programs; therefore, you do not need declarations of Language Environment callable services for programs written in COBOL.
4. CEEIBMxx parts support PL/I applications.
5. CEEIGZxx parts support COBOL applications.

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

Table 19. Imbedding Files in Your Routines

<table>
<thead>
<tr>
<th>To imbed a file in a...</th>
<th>Use statement...</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ routine</td>
<td>#include</td>
</tr>
<tr>
<td>COBOL program</td>
<td>COPY</td>
</tr>
<tr>
<td>PL/I routine</td>
<td>%include</td>
</tr>
</tbody>
</table>

Sample programs

Sample C/C++, COBOL, and PL/I programs are provided for most of the callable services described in this chapter. No sample Fortran programs are provided because you cannot invoke Language Environment callable services from Fortran programs. However, you can use other languages to perform these services on behalf of a Fortran program. These sample programs are also provided online with the Language Environment product for your convenience. On MVS, the sample routines are located in the SCEESAMP partitioned data set.

C/C++ syntax

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

```
Syntax

ceexxx(parml, parm2, ...parmn, fc);
```

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

Use the following syntax to invoke callable services with an omitted feedback code parameter:
Language Environment callable services always have a return type of `void` and should be prototyped as such.

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

You can use the SCEEH.H file `leawi.h` to declare Language Environment callable services, in conjunction with a C/C++ call to a Language Environment callable service, as shown in [Figure 7](#).

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

**Figure 7. Sample Callable Services Invocation Syntax for C/C++**

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

**COBOL syntax**

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

```cobol
CALL "CEExxx" USING—parm1, parm2, ...parmn, fc—;
```

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

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

See [“Parameter list for invoking callable services” on page 117](#) for a description of this syntax.
You can use the SCEESAMP file CEEIGZCT to declare symbolic Language Environment feedback codes, in conjunction with a COBOL call to a Language Environment callable service to return a feedback code, as shown in Figure 8.

COPY CEEIZCT.
CALL "CEExxxx" USING parm1, parm2, ... parmn, fc

Figure 8. Sample Callable Services Invocation Syntax for COBOL

COBOL for MVS & VM Release 2 or later also allows you to omit arguments. In place of the argument, use the OMITTED parameter, as shown below. See Figure 3 on page 112 and Figure 5 on page 112 for an example of omitting parameters in COBOL.

PL/I syntax

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

CALL CEExxxx parm1, parm2, ... parmn, OMITTED;

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

CALL CEExxxx (*, *, ..., *);

Note that you cannot invoke callable services as function references.

See "Parameter list for invoking callable services" on page 117 for a description of this syntax.

The value of register 15 upon return from any Language Environment callable service is undefined. Use of OPTIONS(RETCODE) is not recommended, because a subsequent use of the PLIRETV built-in function returns an undefined value.
If you code your own declarations for the Language Environment callable services, be sure to specify all required arguments in the CALL statement. Figure 4 on page 112 illustrates a call in which the feedback code to CEEDATE has been incorrectly omitted.

You can use the SCEESAMP file CEEIBMAW to declare Language Environment callable services, in conjunction with a PL/I call to a Language Environment callable service, as Figure 9 shows.

```plaintext
%INCLUDE CEEIBMAW;
CALL CEExxx(parm1, parm2, ... parmN, fc);
```

Figure 9. Sample Callable Services Invocation Syntax for PL/I

**Parameter list for invoking callable services**

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

**CEExxx**

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 113 for these file names.

**parm1, parm2, ... parmN**

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

**fc**

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

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

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

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

**Data type definitions**

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

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

Table 20 on page 118, Table 21 on page 119, and Table 22 on page 120 contain data type definitions and their descriptions for C/C++, COBOL, and PL/I, respectively.
Table 20. Data Type Definitions for C/C++

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>C/C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT2</td>
<td>A 2-byte signed integer</td>
<td>signed short</td>
</tr>
<tr>
<td>INT4</td>
<td>A 4-byte signed integer</td>
<td>signed int</td>
</tr>
<tr>
<td>FLOAT4</td>
<td>A 4-byte single-precision floating-point number</td>
<td>float</td>
</tr>
<tr>
<td>FLOAT8</td>
<td>An 8-byte double-precision floating-point number</td>
<td>double</td>
</tr>
<tr>
<td>FLOAT16</td>
<td>A 16-byte extended-precision floating-point number</td>
<td>long double</td>
</tr>
<tr>
<td>COMPLEX8</td>
<td>Short floating-point complex hex number: an 8-byte complex number, whose</td>
<td>Not available</td>
</tr>
<tr>
<td></td>
<td>real and imaginary parts are each 4-byte single-precision floating-point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>numbers</td>
<td></td>
</tr>
<tr>
<td>COMPLEX16</td>
<td>Long floating-point complex hex number: a 16-byte complex number, whose</td>
<td>Not available</td>
</tr>
<tr>
<td></td>
<td>real and imaginary parts are each 8-byte double-precision floating-point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>numbers</td>
<td></td>
</tr>
<tr>
<td>COMPLEX32</td>
<td>Extended floating-point hex number: a 32-byte complex number, whose real</td>
<td>Not available</td>
</tr>
<tr>
<td></td>
<td>and imaginary parts are each 16-byte extended-precision floating-point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>numbers</td>
<td></td>
</tr>
<tr>
<td>POINTER</td>
<td>A platform-dependent address pointer</td>
<td>void *</td>
</tr>
<tr>
<td>CHARn</td>
<td>A string (character array) of length n</td>
<td>char[n]</td>
</tr>
<tr>
<td>VSTRING</td>
<td>A halfword length-prefixed character string (for input); fixed-length</td>
<td>struct _VSTRING(</td>
</tr>
<tr>
<td></td>
<td>80-character string (for output)</td>
<td>_INT2 short length;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>char string[1];</td>
</tr>
<tr>
<td></td>
<td></td>
<td>)string-in;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>char string_out[80];</td>
</tr>
<tr>
<td>FEED_BACK</td>
<td>A mapping of the condition token (fc)</td>
<td>Case 1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>typedef struct {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>short tok_sev ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>short tok_msgno ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int tok_case :2,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tok_sever :3,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tok_ctrl :3 ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>char tok_facid[3];</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int tok_isi ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>} _FEEDBACK ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Case 2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>typedef struct {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>short tok_sev ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>short tok_msgno ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int tok_class_code :2,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tok_class_code :3,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tok_ctrl :3 ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>char tok_facid[3];</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int tok_isi ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>} _FEEDBACK ;</td>
</tr>
</tbody>
</table>
Table 20. Data Type Definitions for C/C++ (continued)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>C/C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE_ENTRY</td>
<td>An HLL-dependent entry constant</td>
<td>FUNCTION POINTER</td>
</tr>
</tbody>
</table>

COBOL data type definitions

Table 21. Data Type Definitions for COBOL

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT2</td>
<td>A 2-byte signed integer</td>
<td>PIC S9(4) USAGE IS BINARY</td>
</tr>
<tr>
<td>INT4</td>
<td>A 4-byte signed integer</td>
<td>PIC S9(9) USAGE IS BINARY</td>
</tr>
<tr>
<td>FLOAT4</td>
<td>A 4-byte single-precision floating-point number</td>
<td>COMP-1</td>
</tr>
<tr>
<td>FLOAT8</td>
<td>An 8-byte double-precision floating-point number</td>
<td>COMP-2</td>
</tr>
<tr>
<td>FLOAT16</td>
<td>A 16-byte extended-precision floating-point number</td>
<td>Not available</td>
</tr>
<tr>
<td>COMPLEX8</td>
<td>Short floating-point complex hex number: an 8-byte complex number,</td>
<td>Not available</td>
</tr>
<tr>
<td></td>
<td>whose real and imaginary parts are each 4-byte single-precision floating-point numbers</td>
<td></td>
</tr>
<tr>
<td>COMPLEX16</td>
<td>Long floating-point complex hex number: a 16-byte complex number,</td>
<td>Not available</td>
</tr>
<tr>
<td></td>
<td>whose real and imaginary parts are each 8-byte double-precision floating-point numbers</td>
<td></td>
</tr>
<tr>
<td>COMPLEX32</td>
<td>Extended floating-point complex hex number: a 32-byte complex number,</td>
<td>Not available</td>
</tr>
<tr>
<td></td>
<td>whose real and imaginary parts are each 16-byte extended-precision floating-point numbers</td>
<td></td>
</tr>
<tr>
<td>POINTER</td>
<td>A platform-dependent address pointer</td>
<td>USAGE IS POINTER</td>
</tr>
<tr>
<td>CHARn</td>
<td>A string (character array) of length n</td>
<td>PIC X(n)</td>
</tr>
<tr>
<td>VSTRING</td>
<td>A halfword length-prefixed character string (for input); fixed-length</td>
<td>01 STRING-IN.</td>
</tr>
<tr>
<td></td>
<td>80-character string (for output)</td>
<td>02 LEN PIC S9(4) USAGE IS BINARY.</td>
</tr>
<tr>
<td></td>
<td>02 TXT PIC X(N).</td>
<td>01 STRING-OUT PIC X(80).</td>
</tr>
<tr>
<td>FEED_BACK</td>
<td>A mapping of the condition token (fc)</td>
<td>Case 1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 FC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 SEV PIC S9(4) USAGE IS BINARY.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 MSGNO PIC S9(4) USAGE IS BINARY.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 FLGS PIC X(1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 FACID PIC X(3).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 ISI PIC X(4).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Case 2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 FC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 CLASS-CODE PIC 9(4) USAGE IS BINARY.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 CAUSE-CODE PIC 9(4) USAGE IS BINARY.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 FLGS PIC X(1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 FACID PIC X(3).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 ISI PIC X(4).</td>
</tr>
<tr>
<td>CEE_ENTRY</td>
<td>An HLL-dependent entry constant</td>
<td>USAGE IS PROCEDURE-POINTER</td>
</tr>
</tbody>
</table>

Table 20. Data Type Definitions for C/C++ (continued)
### PL/I data type definitions

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

#### Table 22. Data Type Definitions for PL/I

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>PL/I</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT2</td>
<td>A 2-byte signed integer</td>
<td>REAL FIXED BINARY (15,0)</td>
</tr>
<tr>
<td>INT4</td>
<td>A 4-byte signed integer</td>
<td>REAL FIXED BINARY (31,0)</td>
</tr>
<tr>
<td>FLOAT4</td>
<td>A 4-byte single-precision floating-point number</td>
<td>REAL FLOAT BINARY (21) or REAL FLOAT DECIMAL (6)</td>
</tr>
<tr>
<td>FLOAT8</td>
<td>An 8-byte double-precision floating-point number</td>
<td>REAL FLOAT BINARY (53) or REAL FLOAT DECIMAL (16)</td>
</tr>
<tr>
<td>FLOAT16</td>
<td>A 16-byte extended-precision floating-point number</td>
<td>REAL FLOAT DECIMAL (33) or REAL FLOAT BINARY (109)</td>
</tr>
<tr>
<td>COMPLEX8</td>
<td>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</td>
<td>COMPLEX FLOAT DECIMAL (6)</td>
</tr>
<tr>
<td>COMPLEX16</td>
<td>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</td>
<td>COMPLEX FLOAT DECIMAL (16)</td>
</tr>
<tr>
<td>COMPLEX32</td>
<td>Extended floating-point hex number: a 32-byte complex number, whose real and imaginary parts are each 16-byte extended-precision floating-point numbers</td>
<td>COMPLEX FLOAT DECIMAL (33)</td>
</tr>
<tr>
<td>POINTER</td>
<td>A platform-dependent address pointer</td>
<td>POINTER</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>A string (character array) of length n</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>VSTRING</td>
<td>A halfword length-prefixed character string (for input); fixed-length 80-character string (for output)</td>
<td>DCL string_in CHAR(n) VARYING; DCL string_out CHAR(80);</td>
</tr>
<tr>
<td>FEED_BACK</td>
<td>A mapping of the condition token (fc)</td>
<td>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);</td>
</tr>
</tbody>
</table>
Callable services

Following are the Language Environment callable services and examples of how to use them in C/C++, COBOL, and PL/I.

**CEE3ABD—Terminate enclave with an abend**

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

**Syntax**

```
CEE3ABD((abcode, clean-up))
```

**abcode (input)**

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

**clean-up (input)**

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

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

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

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

If clean-up is 0, Language Environment condition handling is disabled for the current enclave and termination activities are not performed. Event handlers are not driven; Debug Tool is not invoked; user exits are not invoked; and user-written condition handlers are not invoked.
When clean-up is 1, the abend is processed in the same manner as if it were a non-Language Environment abend. Its processing is affected by the ABPERC and TRAP options, the filedef abends percolated in the assembler user exit, and other elements of the environment related to abend processing. In particular, the condition handler can intercept the abend and give the application a chance to handle the abend. If the condition remains unhandled, normal termination activities are performed: information such as a Language Environment dump is produced, depending on the setting of the TERMTHDACT option; event handlers are driven; Debug Tool is invoked; and user exits are invoked. Assembler user exit settings control if the application actually terminates with an abend.

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

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

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

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

**z/OS UNIX considerations**

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

**Usage notes**

1. **Recommendation:** Language Environment abend codes are usually in the range of 4000 (X'FA0') to 4095 (X'FFF'). You should use the range of 0 to 3999 to avoid confusion with Language Environment abend codes. The value specified for the abend code is passed directly to the ABEND macro without further verification.

2. When TRAP(OFF) is specified, CEE3ABD behaves in a similar manner to clean-up 0.

3. In a non-CICS environment, a system dump of the user address space is taken only if a SYSMDUMP, SYSUDUMP, or SYSABEND DD card is present.

4. Users can look at CEE3AB2 to include user reason codes separately from abend codes.

**For more information**

- For more information about the ABPERC run-time option, see “ABPERC” on page 9.
- For more information about the TRAP run-time option, see “TRAP” on page 93.
- For more information about the TERMTHDACT run-time option, see “TERMTHDACT” on page 78.
- For more information about the CEE3AB2 callable service, see “CEE3AB2—Terminate enclave with an abend and reason code” on page 124.

**Examples**

Figure 10 on page 123 is an example of CEE3ABD called by C/C++.
Figure 10. C/C++ Example of CEE3ABD

Figure 11. COBOL Example of CEE3ABD
CEE3ABD

Figure 12 is an example of CEE3ABD called by PL/I.

```plaintext
*PROCESS MACRO;  
/*****************************/  
/* Licensed Materials - Property of IBM */  
/* 5688-198 (C) Copyright IBM Corp. 1993, 1995 */  
/* All rights reserved */  
/* US Government Users Restricted Rights - Use, */  
/* duplication or disclosure restricted by GSA */  
/* ADP Schedule Contract with IBM Corp. */  
/* */  
/*****************************/  
/*Module/File Name: IBM3ABD  
/*****************************/  
/** Function: CEE3ABD - terminate enclave with an abend  
/**  
/** This example, CEE3ABD is called with a  
/** timing value of 1. This requests an abend that  
/** is deferred until clean-up takes place.  
/** */  
/*****************************/  
PLI3ABD: PROC OPTIONS(MAIN);  
%INCLUDE CEEIBMAM;  
%INCLUDE CEEIBMCT;  
DCL ABDCODE REAL FIXED BINARY(31,0);  
DCL TIMING REAL FIXED BINARY(31,0);  
ABDCODE = 3333;  
TIMING = 1;  
CALL CEE3ABD ( ABDCODE, TIMING );  
END PLI3ABD;  
```

Figure 12. PL/I Example of CEE3ABD

**CEE3AB2—Terminate enclave with an abend and reason code**

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

**Syntax**

```plaintext
>>CEE3AB2(--abcode--,--reasoncode--,--clean-up--)  
```

*abcode (input)*

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

*reasoncode (input)*

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

Indicates if the abend should result in clean-up of the enclave’s resources. The acceptable values for `clean-up` are as follows:

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

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

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

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

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

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

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

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

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

**z/OS UNIX consideration**

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

For more information
- For more information about the ABPERC run-time option, see "ABPERC" on page 9.
- For more information about the TRAP run-time option, see "TRAP" on page 93.
- For more information about the TERMTHDACT run-time option, see "TERMTHDACT" on page 78.
- For more information about the ABEND macro, see z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO.
- For more information about the CEE3ABD callable service, see "CEE3ABD—Terminate enclave with an abend" on page 121.

Examples
**Figure 13** is an example of CEE3AB2 called by C/C++.

```c
/*Module/File Name: EDC3AB2 */
/*******************************************************************************/
/**/ /* THIS EXAMPLE CALLS CEE3AB2 TO TERMINATE THE ENCLAVE WITH AN ABEND */
/**/ /* AFTER CLEAN-UP TAKES PLACE. USER COULD ALSO SPECIFY THEIR OWN */
/**/ /* REASON CODE. */
/**/ /* */
/*******************************************************************************/
#include <leawi.h>
int main(void) {

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

**Figure 13. C/C++ Example of CEE3AB2**

**Figure 14** on page 127 is an example of CEE3AB2 called by COBOL.
Figure 14. COBOL Example of CEE3AB2

**Module/File Name: IGTAB2**

**Function:** CEE3AB2 - Terminate enclave **
** with an abend and user defined **
** reason code **

**In this example, CEE3AB2 is called to**
** terminate the enclave with an abend and**
** a user defined reason code along with**
** Language Environment cleanup**

IDENTIFICATION DIVISION.
PROGRAM-ID. CEE3AB2.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ABCODE PIC 9(9) BINARY.
01 RESCODE PIC 9(9) BINARY.
01 TIMING PIC 9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLMGET.

**1234 is the abend code to be issued, **
** 9 is the reason code for the abend. **
** A timing of four requests an abend **
** with clean-up. **

MOVE 1234 TO ABCODE.
MOVE 9 TO RESCODE.
MOVE 4 TO TIMING.
CALL "CEE3AB2" USING ABCODE, RESCODE, TIMING.
GOBACK.

Figure 15 on page 128 is an example of CEE3AB2 called by PL/I.
CEE3CIB

**CEE3CIB—Return pointer to condition information block**

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

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

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

```c
cee3cib(cond_token, cib_ptr, fc)
```

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

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

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the results of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE35S</td>
<td>1</td>
<td>3260</td>
<td>No condition was active when a call to a condition management routine was made.</td>
</tr>
<tr>
<td>CEE35U</td>
<td>1</td>
<td>3262</td>
<td>An invalid condition token was passed. The condition token did not represent an active condition.</td>
</tr>
</tbody>
</table>

**Usage notes**

- Because the CIB is used only for synchronous signals, you should not use CEE3CIB in signal catchers that are driven for asynchronous signals.
- After the condition handling functions return control to your application, `cib_ptr` is no longer valid.
- z/OS UNIX consideration—In multithread applications, CEE3CIB returns the CIB associated with the current token on only the current thread.

**For more information**

- For more information about the CEE3CIB callable service, see [z/OS Language Environment Programming Guide](#).

**Examples**

[Figure 16 on page 130](#) is an example of CEE3CIB called by C/C++. 
/*Module/File Name: EDC3CIB */
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>
#ifdef __cplusplus
extern "C" {
#endif

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
_FEEDBACK *newfc) {

    _CEECIB *cib_ptr;
    _FEEDBACK cibfc;
    CEE3CIB(fc, &cib_ptr, &cibfc);
    
    /* verify that CEE3CIB was successful */
    if ( _FBCHECK ( cibfc , CEE000 ) != 0 ) {
        printf("CEE3CIB failed with message number %d\n", cibfc.tok_msgno);
        exit(2999);
    }
    printf("%s \n", (*cib_ptr).cib_eye);
    printf("%d \n", cib_ptr->cib_cond.tok_msgno);
    printf("%s \n", cib_ptr->cib_cond.tok_facid);
    *result = 10;
}
#endif

int main(void) {
    _FEEDBACK fc;
    _ENTRY routine;
    _INT4 token;
    Int x, y, z;

    /* set the routine structure to point to the handler */
    /* and use CEEHDLR to register the user handler */

    token = 99;
    routine.address = (_POINTER)&handler;;
    routine.nesting = NULL;
    CEEHDLR($routine, &token, &fc);
    /* verify that CEEHDLR was successful */
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }
    
    x = 5;
    y = 0;
    z = x / y;
}

/***************************************************************************/
/* handler is a user condition handler */
/***************************************************************************/

Figure 16. C/C++ Example of CEE3CIB

Figure 17 on page 131 is an example of CEE3CIB called by COBOL.
Figure 17. COBOL Example of CEE3CIB

Figure 18 on page 132 is an example of CEE3CIB called by PL/I.
CEE3CTY

CEE3CTY—Set default country

CEE3CTY sets the default country. A calling routine can change or query the current national country setting. The country setting affects the date format, the time format, the currency symbol, the decimal separator, and the thousands separator.

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

CEE3CTY also affects the default symbols for the NLS and date and time callable services.

CEE3CTY affects only the Language Environment NLS and date and time services, not the Language Environment locale callable services or C locale-sensitive functions.

The current default country setting, as well as previous default settings that have not been discarded, are stored on the stack on a LIFO (last in, first out) basis. The current default country setting is always on the top of the stack.
There are two methods of changing the default country setting with CEE3CTY:

- Specify the new default country setting and place it on top of the stack using a `function` value of 1 (SET). This discards the previous default setting.
- Specify the new default country setting and place it on top of the stack using a `function` value of 3 (PUSH). This pushes the previous default setting down on the stack so that you can later retrieve it by discarding the current setting.

To illustrate the second method, suppose you live in the United States and the code for the United States is specified as the default at installation. If you want to use the French defaults for a certain application, you can use CEE3CTY to PUSH France as the default country setting; then when you want the defaults for the United States, you can POP France from the top of the stack, making the United States the default setting.

### Syntax

```
CEE3CTY(function, country_code, fc)
```

#### function (input)

A fullword binary integer specifying the service to be performed. The possible values for `function` are:

- **1**—**SET** Establishes the `country_code` parameter as the current country. The top of the stack is, in effect, replaced with `country_code`.
- **2**—**QUERY** Returns the current country code on top of the stack to the calling routine. The current code is returned in the `country_code` parameter.
- **3**—**PUSH** Pushes the `country_code` parameter onto the top of the country code stack, making it the current country code. Previous country codes on the stack are retained on a LIFO basis, which makes it possible to return to a prior country code at a later time.
- **4**—**POP** Pops the current country code. The last country code that was affected by a PUSH now becomes the current `country_code`. On return to the calling routine, the `country_code` parameter contains the discarded country code. If the stack contains only one country code, the code cannot be popped because the stack would be empty after the call. Therefore, no action is taken and a feedback code indicating such is returned to the calling routine.

#### country_code (input/output)

A 2-character fixed-length string. `country_code` is not case-sensitive. Table 32 on page 515 contains a list of valid country codes. It is used in the following ways for the different `functions`:

<table>
<thead>
<tr>
<th>If <code>function</code> is:</th>
<th>then <code>country_code</code>:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 or 3</td>
<td>Contains the desired 2-character country code. In this case, it is an input parameter. Table 32 on page 515 contains a list of valid country codes.</td>
</tr>
<tr>
<td>2</td>
<td>Returns the current 2-character country code on top of the stack. In this case, it is an output parameter.</td>
</tr>
</tbody>
</table>
If function is: then country_code:

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns the discarded 2-character country code. In this case, it is an output parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc (output)</td>
<td>A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3BV</td>
<td>2</td>
<td>3455</td>
<td>Only one country code was on the stack when a POP request was made to CEE3CY. The current country code was returned in the country-code.</td>
</tr>
<tr>
<td>CEE3C0</td>
<td>3</td>
<td>3456</td>
<td>The country code country-code for the PUSH or SET function for CEE3CY was invalid. No operation was performed.</td>
</tr>
<tr>
<td>CEE3C1</td>
<td>3</td>
<td>3457</td>
<td>The function function specified for CEE3CY was not recognized. No operation was performed.</td>
</tr>
</tbody>
</table>

Usage notes

- The default setting for country_code is already on the stack when you start a program. You do not have to PUSH it there.
- The bytes X'0E' and X'0F' representing shift-out and shift-in codes are not affected by any country_code setting.
- C/C++ consideration—Language Environment provides locales used in C and C++ to establish default formats for the locale-sensitive functions and locale callable services, such as date and time formatting, sorting, and currency symbols. To change the locale, you can use the setlocale() library function or the CEESETL callable service.

The settings of CEESETL or setlocale() do not affect the setting of the CEE3CTY callable service. CEE3CTY affects only Language Environment NLS and date and time services. setlocale() and CEESETL affect only C/C++ locale-sensitive functions and Language Environment locale callable services.

To ensure that all settings are correct for your country, use CEE3CTY and either CEESETL or setlocale().
- z/OS UNIX consideration—CEE3CTY applies to the enclave. Every thread in the enclave has the same country setting.

For more information

- For more information about the RPTOPTS and RPTSTG run-time options, see “RPTOPTS” on page 66 and “RPTSTG” on page 68.
- For a list of the default settings for a specified country, see Table 32 on page 515
- For more information about the COUNTRY run-time option, refer to “COUNTRY” on page 22.
- For more information on setlocale(), see z/OS XL C/C++ Programming Guide

Examples

Figure 19 on page 135 is an example of CEE3CTY called by C/C++. 
Figure 20 on page 136 is an example of CEE3CTY called by COBOL.

Figure 19. C/C++ Example of CEE3CTY

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceedcct.h>

int main(void) {
    _FEEDBACK fc;
    _INT4 function;
    _CHAR2 country;

    /* query the current country setting */
    function = 2; /* function 2 is query */
    CEE3CTY(&function,country,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) { 
        printf("CEE3CTY failed with message number \%d\n", fc.tok_msgno);
        exit(2999);
    }

    /* if the current country is not Canada then set */
    /* it to Canada */
    if (memcmp(country,"CA",2) != 0) {
        memcpy(country,"CA",2);
        function = 1; /* function 1 is set */
        CEE3CTY(&function,country,&fc);
        if ( _FBCHECK ( fc , CEE000 ) != 0 ) { 
            printf("CEE3CTY failed with message number \%d\n", fc.tok_msgno);
            exit(2999);
        }
    }
}
```
IDENTIFICATION DIVISION.

PROGRAM-ID. CBL3CTY.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 FUNCTN PIC S9(9) BINARY.
01 COUNTRY PIC X(2).
01 FC.
02 Condition-Token-Value.
COPY CEEIGZCT.
03 Case-1-Condition-ID.
04 Severity PIC S9(4) BINARY.
04 Msg-No PIC S9(4) BINARY.
03 Case-2-Condition-ID
REDEFINES Case-1-Condition-ID.
04 Class-Code PIC S9(4) BINARY.
04 Cause-Code PIC S9(4) BINARY.
03 Case-5ев-Ctl PIC X.
03 Facility-ID PIC XXX.
02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.

PARA-3CTYQR.

***************************************************************************
** Call CEE3CTY with the QUERY function, and display current country code. **
***************************************************************************
MOVE 2 TO FUNCTN.
CALL "CEE3CTY" USING FUNCTN, COUNTRY, FC.
IF CEE000 of FC THEN
DISPLAY "THE CURRENT COUNTRY CODE IS "
COUNTRY
ELSE
DISPLAY "CEE3CTY(query) failed with msg "
Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.

PARA-3CTYSET.

***************************************************************************
** If the current country code is not the US, call CEE3CTY with the SET function to make **
** it the US. Display result. **
***************************************************************************
IF ( COUNTRY IS NOT = "US" ) THEN
MOVE 1 TO FUNCTN
MOVE "US" TO COUNTRY
CALL "CEE3CTY" USING FUNCTN, COUNTRY, FC
IF CEE000 of FC THEN
DISPLAY "THE NEW COUNTRY CODE IS ",
COUNTRY
ELSE
DISPLAY "CEE3CTY(set) failed with msg "
Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.
END-IF.

GOBACK.

Figure 20. COBOL Example of CEE3CTY
Figure 21 is an example of CEE3CTY called by PL/I.

```pli
*PROCESS MACRO;
/* Module/File Name: IBM3CTY */
/*****************************************************************************/
/** */
/** Function: CEE3CTY - set current country */
/** */
/** In this example, a call is made to the query */
/** function of CEE3CTY to return the current */
/** default country setting. This is then */
/** printed out. If the current country code is */
/** not 'US', then it is set to 'US' and printed. */
/** */
/*****************************************************************************/
PL3CTY: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;
DCL FUNCTN REAL FIXED BINARY(31,0);
DCL COUNTRY CHARACTER (2);
DCL 01 FC, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
   05 Case BIT(2),
   05 Severity BIT(3),
   05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
   REAL FIXED BINARY(31,0);
FUNCTN = 2; /* Specify 2 for the query function */
/* Call CEE3CTY with the query function to */
/* return the current country setting */
CALL CEE3CTY ( FUNCTN, COUNTRY, FC );
/* If CEE3CTY ran successfully, print the */
/* current country */
IF FBCHECK( FC, CEE000) THEN DO;
   PUT SKIP LIST( 'The current country code is ' ||
                  COUNTRY || '.' );
   END;
ELSE DO;
   DISPLAY( 'CEE3CTY failed with msg ' ||
            FC.MsgNo );
   STOP;
   END;
```

Figure 21. PL/I Example of CEE3CTY (Part 1 of 2)
CEE3DLY—Suspend processing of the active enclave in seconds

CEE3DLY provides a service for Language Environment-conforming applications that suspends the processing of the active enclave for a specified number of seconds. The maximum is 1 hour.

**Syntax**

```
CEE3DLY(input_seconds, fc)
```

**input_seconds**
A full-word binary value in the range of 0 to 3600 that specifies the total number of seconds during which the enclave should be suspended.

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE3QQ</td>
<td>1</td>
<td>CEE3930W</td>
<td>The input value <code>input_value</code> in a call to the callable service <code>callable_service_name</code> was not within the valid range.</td>
</tr>
<tr>
<td>CEE3QS</td>
<td>1</td>
<td>CEE3932W</td>
<td>The system service <code>system_service</code> failed with return code <code>return_code</code> and reason code <code>reason_code</code>.</td>
</tr>
</tbody>
</table>

**Usage notes**

- CICS consideration—CEE3DLY is available under CICS.
- z/OS UNIX consideration—CEE3DLY is handled by the z/OS UNIX System Services when POSIX is set to ON.
- This service is not intended for timing requests. Delays up to the nearest second might occur in some circumstances.
- In a multi-threaded application, only the calling thread is suspended.
Examples

Figure 22 is an example of CEE3DLY called by C/C++.

```c
/*Module/File Name: ED3DLY */
*******************************************************************************/
/**/
/* THIS EXAMPLE CALLS CEE3DLY TO SUSPEND PROCESSING OF THE ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF SECONDS. */
/**/
*******************************************************************************/
#include <time.h>
#include <stdio.h>
#include <leawi.h>
#include <ceeedccct.h>

int main(void)
{
    _INT4 seconds;
    _FEEDBACK fc;
    time_t ltime;
    /* Get the time in seconds */
    time(&ltime);
    printf("CEE3DLY Start time : %s", ctime(&ltime));
    seconds = 10;
    CEE3DLY(&seconds,&fc);
    if ( _FBCHECK ( fc , CEE000 ) !=0 ) {
        printf("CEE3DLY failed with message number %d\n",fc.tok_msgno);
        exit(999);
    }
    time(&ltime);
    printf("CEE3DLY Finish time : %s", ctime(&ltime));
}
```

Figure 23. C/C++ Example of CEE3DLY

Figure 23 is an example of CEE3DLY called by COBOL.

```cobol
CBL LIB,QUOTE
******************************************
** IBM Language Environment **
** Licensed Materials - Property of IBM **
** 5694-A01 **
** (C) Copyright IBM Corp. 2006 **
** All Rights Reserved **
** US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA **
** ADP Schedule Contract with IBM Corp. **
******************************************
MODULE/FILE NAME: IGZT3DLY
*******************************************************************************/
** FUNCTION: CEE3DLY - SUSPEND ENCLAVE EXECUTION IN SECONDS **
** THIS EXAMPLE CALLS CEE3DLY TO SUSPEND PROCESSING OF THE ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF SECONDS. **
*******************************************************************************/
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3DLY.
DATA DIVISION.
```

Figure 23. COBOL Example of CEE3DLY (Part 1 of 2)
Figure 24 on page 141 is an example of CEE3DLY called by PL/I.
Figure 24. PL/I Example of CEE3DLY (Part 1 of 2)
CEE3DMP—Generate dump

CEE3DMP generates a dump of Language Environment and the member language libraries. Sections of the dump are selectively included, depending on options specified with the `options` parameter. When an error occurs that would cause a CEEDUMP to be taken, and this is a POSIX application, Language Environment writes this dump to the current directory. Output from CEE3DMP is written to one of the following (in top down order):

- The directory found in `_CEE_DMPTARG`, if found.
- The current working directory, if this is not the root (/), and if the directory is writable, and if the dump pathname (made up of the cwd pathname plus the dump file name) does not exceed 1024 characters.
- The directory found in environment variable TMPDIR (if the temporary directory is not /TMP.
- /TMP

(See the Fname suboption of CEE3DMP on page 147 for more information about the dump output filename.)

If longer than 60 characters, the dump title is truncated to 60 characters in order to match the record size of the dump file. Only nested enclaves within a single process are supported.

CEE3DMP establishes a condition handler that captures all conditions that occur during dump processing. It terminates the section of the dump in progress when a condition occurs and inserts the following line into the dump (`nnnnnnnn` is the instruction address at the time of the exception). After this line is inserted in the report, dump processing continues for other member languages until CEE3DMP is complete.

```
DLYSECS = 30;
CALL CEE3DLY(DLYSECS, FC2);
IF FBCHECK(FC2, CEE000) THEN DO;
  CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC1 );
  IF FBCHECK(FC1, CEE000) THEN DO;
    PUT SKIP LIST ( 'CEE3DLY FINISH DATE AND TIME: ' || GREGORN );
    END;
  ELSE DO;
    PUT ('CEELOCT FAILED WITH MSG ' || FC1.MSGNO );
    STOP;
  END;
END;
ELSE DO;
  PUT SKIP LIST ( 'CEE3DLY IS SUCCESSFUL! ' );
  END;
ELSE DO;
  PUT SKIP LIST ( 'CEE3DLY FAILED WITH MSG ' || FC2.MSGNO );
  STOP;
END;
END PLI3DLY;
```

Figure 24. PL/I Example of CEE3DLY (Part 2 of 2)

Exception occurred during dump processing at nnnnnnnn

If an abend occurs, or if any other condition occurs or is raised, the condition manager attempts to handle it. If the condition remains unhandled, and it is of sufficient severity, the condition manager might, based on if the TERMTHDACT TRACE or DUMP suboption is specified, invoke dump services and then terminate
the program. You do not have to call CEE3DMP to use the dump services. Any routine can use them. To support this case, dump services are invoked as follows:

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

Reprinting of section title and control block name at the top of each page is suppressed. Only the main title CEE3DMP: ... is reprinted.

Syntax

```c
CEE3DMP(---title---,---options---,...fc---)
```

title (input)
An 80-byte fixed-length character string containing a title printed at the top of each page of the dump.

options
A 255-byte fixed-length character string enclosed in single quotes containing options describing the type, format, and destination of dump information. Options are declared as a string of keywords separated by blanks or commas. Some options have suboptions that follow the option keyword and are contained in parentheses. The options can be specified in any order, but the last option declaration is honored if there is a conflict between it and any preceding options. The following options are recognized by Language Environment:

ENCLAVE(ALL | CURRENT | n)
Dumps the current enclave, a fixed number of enclaves, or all enclaves associated with the current process; n is an integer ranging from 1 to $2^{31}-1$, inclusive, that indicates the maximum number of enclaves that should be dumped. ENCLAVE(CURRENT) and ENCLAVE(1) are equivalent.

THREAD(ALL | CURRENT)
Dumps the current thread (the thread that invoked this service) or all threads associated with the current enclave. When you specify THREAD(ALL) and more than one thread is running, the library quiesces all threads before writing the dump. Therefore, the state of the library changes from the time the dump is requested to the time the dump is written.

TRACEback
Includes a traceback of all routines on the call chain. The traceback shows transfers of control from either calls or exceptions. The traceback extends backwards to the main program of the current thread. PL/I transfers of control into BEGIN-END blocks or ON-units are considered calls.

NOTRACEback
Does not include a traceback.

FILES
Includes attributes of all open files and the buffer contents used by the files. The particular attributes displayed are defined by the member
languages. File buffers are dumped when FILE and STORAGE are specified. File control blocks are dumped when FILE and BLOCKS are specified.

**NOFILEs**
Does not include file attributes of open files.

**VARIABLES**
Includes a symbolic dump of all variables, arguments, and registers. Variables include arrays and structures. Register values are those saved in the stack frame at the time of call. There is no way to print a subset of this information. Variables and arguments are printed only if the symbol tables are available. A symbol table is generated when a program is compiled with the options shown below for each HLL, except for PL/I, which does not support the VARIABLE option.

<table>
<thead>
<tr>
<th>Language</th>
<th>Compile-Time Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>TEST(SYM)</td>
</tr>
<tr>
<td>C++</td>
<td>TEST</td>
</tr>
<tr>
<td>COBOL</td>
<td>TEST or TEST(h,SYM)</td>
</tr>
</tbody>
</table>

The variables, arguments, and registers are dumped, beginning with the routine that called CEE3DMP. The dump proceeds up the chain for the number of routines specified by the STACKFRAME option. See below for a description of the STACKFRAME option.

**NOVARIABLES**
Does not include a dump of variables, arguments, and registers.

**BLOCKS**
Dumps the control blocks used in Language Environment and member language libraries. Global control blocks, as well as control blocks associated with routines on the call chain, are printed. Control blocks are printed for the routine that called CEE3DMP. The dump proceeds up the call chain for the number of routines specified by the STACKFRAME option (see below). Control blocks for files are also dumped if the FILES option was specified. See the FILES option above for more information. If the TRACE run-time option is set to ON, the trace table is dumped when BLOCKS is specified. If the heap storage diagnostics report is requested through the HEAPCHK run-time option, the report is displayed when BLOCKS is specified.

**NOBLOCKs**
Suppresses the dump of control blocks.

**STORAGE**
Dumps the storage used by the program. The storage is displayed in hexadecimal and character format. Global storage, as well as storage associated with each routine on the call chain, is printed. Storage is dumped for the routine that called CEE3DMP, which proceeds up the call chain for the number of routines specified by the STACKFRAME option. Storage for all file buffers is also dumped if the FILES option was specified (see above).

**NOSTORAGE**
Suppresses storage dumps.

**REGSTOR**
Controls the amount of storage (reg_stor_amount) that is dumped around registers. reg_stor_amount indicates storage in bytes to be dumped around
each register and must be in the range of 0 to 256. The number is rounded up to the nearest multiple of 32. The default is REGSTOR(96).

Restriction: You must specify REGSTOR(0) as a CEE3DMP option if a dump storage around registers is not required.

StackFrame(n|ALL)
Specifications the number of stack frames dumped from the call chain. If STACKFRAME(ALL) is specified, all stack frames are dumped. No stack frame storage is dumped if STACKFRAME(0) is specified. The particular information dumped for each stack frame depends on the VARIABLE, BLOCK, and STORAGE option declarations specified for CEE3DMP. The first stack frame dumped is the one associated with the routine that called CEE3DMP, followed by its caller, and proceeding backwards up the call chain.

PAGEsize(n)
Specifications the number of lines on each page of the dump. This value must be greater than 9. A value of 0 indicates that there should be no page breaks in the dump. The default setting is PAGESIZE(60). Refer to the CEEDUMP run-time option for an optional method of setting the number of lines on each page of the dump.

FNAME(ddname)
Specifications the ddname of the file to which the dump report is written. The ddname supplied in this option must be a valid ddname for the system on which the application is running. CEE3DMP does not check the ddname for validity, nor does CEE3DMP translate or modify the ddname. Supplying an invalid ddname can result in unpredictable behavior. The default ddname CEEDUMP is used if this option is not specified.

CONDITION
Specifications that for each active condition on the call chain, the following information is dumped from the CIB:
• The address of the CIB.
• The message associated with the current condition token.
• The message associated with the original condition token, if different from the current one.
• The location of the error.
• The machine state at the time the condition manager was invoked, if an abend or hardware condition occurred.
• The abend code and reason code, if the condition occurred as a result of an abend.
• Language-specific error information.

The information supplied by Language Environment-conforming languages differs. PL/I supplies DATAFIELD, ONCHAR, ONCOUNT, ONFILE, ONKEY, and ONSOURCE built-in function (BIF) values, which are shown in the context of the condition raised.

This option does not apply to asynchronous signals.

NOCONDITION
Does not dump condition information for active conditions on the call chain.

ENTRY
Includes in the dump a description of the routine that called CEE3DMP and the contents of the registers on entry to CEE3DMP.
CEE3DMP

**NOENTRY**
Does not include in the dump a description of the routine that called CEE3DMP and the contents of the registers on entry to CEE3DMP.

**GENOpts**
Includes in the dump a run-time options report like that generated by the use of the RPTOPTS run-time option.

**NOGENOpts**
Does not include in the dump a run-time options report like that generated by the use of the RPTOPTS run-time option.

**fc (output)**
A 12-byte feedback token code that indicates the result of a call to CEE3DMP. If specified as an argument, feedback information, in the form of a condition token, is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE317</td>
<td>1</td>
<td>3111</td>
<td>CEDUMP was defined as a dummy data set. No Language Environment dump processing was performed.</td>
</tr>
<tr>
<td>CEE30U</td>
<td>2</td>
<td>3102</td>
<td>Invalid CEE3DMP options or suboptions were found and ignored.</td>
</tr>
<tr>
<td>CEE30V</td>
<td>3</td>
<td>3103</td>
<td>An error occurred in writing messages to the dump file.</td>
</tr>
</tbody>
</table>

**Usage notes**
- Language Environment is sensitive to the national language when it writes dump output.
  - When the national language is upper case U.S. English or Japanese, the environment variable _CEE_UPPERCASE_DATA can be used to determine if variable data in the dump output is in uppercase.
  - When this environment variable is set to YES, variable data (entry point names, program unit names, variable names, Trace Entry in EBCDIC data, hexadecimal/EBCDIC displays of storage) will be in uppercase.
  - When this environment variable is not set or set to a value other than YES, variable data will not be in uppercase. Variable data is never in uppercase when the national language is mixed-case U.S. English.
- CICS consideration—Only ENCLAVE(CURRENT) and ENCLAVE(1) are supported on CICS.
- MVS consideration—On MVS, all values for the ENCLAVE option are supported.
- z/OS UNIX consideration—CEE3DMP applies to the enclave. When you call CEE3DMP in a multithread environment, the current thread or all threads might be dumped. Enclave- and process-related storage (along with storage related to threads other than the current thread) might have changed in value from the time the dump request was issued.
  - If the CEEDUMP DD has a PATH= parameter, the dump is directed to the hierarchical file system (HFS) file specified.
  - If your application is running under z/OS UNIX System Services and is either running in an address space created by using the fork() function or is invoked by one of the exec family of functions, the dump is written to the HFS.
Language Environment writes the dump to a file in your current working
directory, unless that directory is the root directory, in which case the dump is
written to a file in the directory /tmp.
The name of this file changes with each dump and uses the following format:

```
/path/Fname.Date.Time.Pid
```

- `path`  The current working directory
- `Fname` The name specified in the FNAME parameter on the call to CEE3DMP
  (default is CEEDUMP)
- `Date`  The date the dump is taken, appearing in the format YYYYMMDD (such as
  19940325 for March 25, 1994)
- `Time`  The time the dump is taken, appearing in the format HHMMSS (such as
  175501 for 05:55:01 PM)
- `Pid`   The process ID the application is running in when the dump is taken

For more information
- See "TERMTHDACT" on page 78 for more information about the
  TERMTHDACT run-time option.
- For more information on generating dumps, see z/OS Language Environment
  Debugging Guide.

Examples

**Figure 25** is an example of CEE3DMP called by C/C++.

```c
/*Module/File Name: EDC3DMP */
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceeedcct.h>
#define OPT_STR "THREAD(CURRENT) TRACEBACK FILES"
int main(void) {
  _CHAR80 title = "This is the title of the dump report";
  _CHAR255 options;
  FILE *f;
  _FEEDBACK fc;
  memset(options,' ',sizeof(options));
  memcpy(options,OPT_STR,sizeof(OPT_STR)-1);
  f = fopen("my.file","wb");
  fprintf(f,"my file record 1\n");
  CEE3DMP(title,options,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEE3DMP failed with msgno %d\n",
            fc.tok_msgno);
    exit(2999);
  }
}
```

**Figure 25. C/C++ Example of CEE3DMP**

**Figure 26 on page 148** is an example of CEE3DMP called by COBOL.
Figure 26. COBOL Example of CEE3DMP

Figure 27 on page 149 is an example of CEE3DMP called by PL/I.

CBL LIB,QUOTE
*Module/File Name: IGZT3DMP
*******************************************************************************
** **
** CBL3DMP - Call CEE3DMP to generate a dump **
** **
** In this example, a call to CEE3DMP is made **
** to request a dump of the run-time **
** environment. Several options are specified **
** to customize the dump. **
*******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3DMP.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DMPTITL PIC X(80).
01 OPTIONS PIC X(255).
01 FC.
  02 Condition-Token-Value.
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
*******************************************************************************
** Specify title to appear on each page of the **
** dump report. **
** Specify options that will request that a **
** traceback be provided, but no variables, **
** stack frames, condition information, or **
** registers be dumped. **
*******************************************************************************
PARA-CBL3DMP.
  MOVE "This is the dump report title." TO DMPTITL.
  MOVE "TRACE NOVAR SF(0) NOCOND NOENTRY" TO OPTIONS.
  CALL "CEE3DMP" USING DMPTITL, OPTIONS, FC.
  IF NOT CEE000 of FC THEN
    DISPLAY "CEE3DMP failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
GOBACK.
CEE3GRC—Get the enclave return code

CEE3GRC retrieves the current value of the user enclave return code. Use CEE3GRC in conjunction with CEE3SRC to get and then set user enclave return codes.

Syntax

```plaintext
CEE3GRC(---return_code---,---fc---)
```

`return_code (output)`

The enclave return code.
**CEE3GRC**

*fc (output)*

A feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**Usage notes**

- z/OS UNIX consideration—CEE3GRC is not supported in multithread applications.
- PL/I MTF consideration—CEE3GRC is not supported in PL/I MTF applications.
- PL/I consideration—When running PL/I with POSIX(ON), CEE3GRC is not supported.

**For more information**

- See [“CEE3SRC—Set the enclave return code” on page 212](#) for more information about the CEE3SRC callable service.
- See [z/OS Language Environment Programming Guide](#) for more information about the CEE3GRC and CEE3SRC callable services.

**Examples**

[Figure 28 on page 151](#) is an example of a C/C++ main() routine that calls CEEHDLR and CEE3GRC.
Figure 29 on page 152 is an example of a C/C++ user-written condition handler that sets a user enclave return code.
Figure 30 is an example of a C/C++ subroutine that generates the divide-by-zero condition.

```c
/*Module/File Name: EDC3SRC */
/**************************************************************/
/** */
/** Function: CEE3SRC - Set the enclave return code. */
/** */
/** This is the user-written condition handler */
/** registered by CEGETRC. It invokes CEE3SRC to set */
/** the enclave return code to 999998 */
/** when a divide-by-zero condition is encountered. */
/** */
/**************************************************************/
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>
#define RESUME 10
#define PERCOLATE 20
/**************************************************************/

#ifdef __cplusplus
extern "C" void CESETRC (_FEEDBACK *, _INT4 *,
_INT4 *, _FEEDBACK *);
#endif

void CESETRC (_FEEDBACK *cond, _INT4 *input_token,
_INT4 *result, _FEEDBACK *new_cond)
{
_INT4 enclave_RC;
_FEEDBACK feedback;
if ( _FBCHECK ( *cond , CEE349 ) == 0 )
{
  enclave_RC = 999998;
  CEE3SRC(&enclave_RC, &feedback);
  *result = RESUME;
}
else
  *result = PERCOLATE;
}

Figure 29. C/C++ User-Written Condition Handler that Sets a User Enclave Return Code

Figure 30 is an example of a C/C++ subroutine that generates the divide-by-zero condition.

```c
/*Module/File Name: EDCDIV */
#include <stdio.h>
#include <string.h>
/**************************************************************/
/** */
/* This is a divide-by-zero routine. It divides */
/* an input integer by a constant. */
/** */
/**************************************************************/

void CERCDIV (int Integer)
{
  int num;
  num = 1/Integer;
}

Figure 30. C/C++ Subroutine that Generates the Divide-by-Zero Condition

```
**Module/File Name: IGZT3GRC
**************************************************
** CBL2GRC - Call the following Lang. Env. svcs: **
** : CEEHDLR - register user condition **
** handler **
** : CEE3GRC - get enclave return code **
**
** 1. Registers user condition handler CESETRC. **
** 2. Program then calls CERCDIV which performs **
** a divide by zero operation. **
** 3. CESETRC gets control and set the enclave **
** return code to 999998 and resumes. **
** 4. Regains control and retrieves the enclave **
** return code. **
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3GRC.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 TOKEN PIC X(4).
01 IDIVISOR PIC S9(9) BINARY VALUE ZERO.
01 ENCLAVE-RC PIC S9(9) BINARY.
**
** Declares for condition handling
**
01 PGMPTR USAGE IS PROCEDURE-POINTER.
01 FBCODE.
02 Condition-Token-Value.
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
  04 Severity PIC S9(4) BINARY.
  04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
  REDEFINES Case-1-Condition-ID.
  04 Class-Code PIC S9(4) BINARY.
  04 Cause-Code PIC S9(4) BINARY.
  03 Case-Seq-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
END-IF.
**************************************************
** Call CERCDIV to cause a divide by zero **
** condition
**************************************************
CALL "CERCDIV" USING IDIVISOR.

Figure 31. COBOL Main Routine that Calls CEEHDLR and CEE3GRC (Part 1 of 2)
Figure 32 is an example of a COBOL condition handler that sets a user enclave return code and resumes when a divide-by-zero condition occurs.

```
CBL C,RENT,OPTIMIZE,NODYNAM,LIB,QUOTE
*Module/File Name: IGZT3SRC
*************************************************
** DRV3SRC - Drive sample program for CEE3SRC. **
** *************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRV3SRC.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 ROUTINE PROCEDURE-POINTER.
 01 DENOMINATOR PIC S9(9) BINARY.
 01 NUMERATOR PIC S9(9) BINARY.
 01 RATIO PIC S9(9) BINARY.
 01 TOKEN PIC S9(9) BINARY VALUE 0.
 01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
REGISTER-HANDLER.
*************************************************
** Register handler
*************************************************
SET ROUTINE TO ENTRY "CBL3SRC".
CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
  DISPLAY "CEEHDLR failed with msg "
  Msg-No of FC UPON CONSOLE
END-IF.
```

Figure 32. COBOL Condition Handler that Sets a User Enclave Return Code and Resumes when a Divide-by-Zero Condition Occurs (Part 1 of 3)
RAISE-CONDITION.
**************************************************
** Cause a zero-divide condition.
**************************************************
MOVE 0 TO DENOMINATOR.
MOVE 1 TO NUMERATOR.
DIVIDE NUMERATOR BY DENOMINATOR
GIVING RATIO.

UNREGISTER-HANDLER.
**************************************************
** Unregister handler
**************************************************
CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
  DISPLAY "CEEHDLU failed with msg "
  Msg-No of FC UPON CONSOLE
END-IF.
STOP RUN.
END PROGRAM DRV3SRC.

**************************************************
** CBL3SRC - Call CEE3SRC to set the enclave
** return code
**
** This is an example of a user-written
** condition handler that sets a user
** enclave return code and resumes when
** a divide-by-zero condition occurs.
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3SRC.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ENCLAVE-RC PIC S9(9) BINARY.
  01 FEEDBACK.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
     03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
     03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
     03 Case-Sev-Ctl
      04 I-S-Info PIC S9(9) BINARY.
     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.
  08 RESUME VALUE +10.
  08 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.

Figure 32. COBOL Condition Handler that Sets a User Enclave Return Code and Resumes
when a Divide-by-Zero Condition Occurs (Part 2 of 3)
Figure 33 on page 157 is an example of a COBOL subroutine that generates a divide-by-zero condition.
**Figure 33. COBOL Subroutine that Generates a Divide-by-Zero**

**Figure 34** is an example in PL/I that sets and retrieves the user enclave return code when a divide-by-zero is generated.
CEE3GRN—Get name of routine that incurred condition

CEE3GRN gets the name of the most current Language Environment-conforming routine where a condition occurred. If there are nested conditions, the most recently signaled condition is used.

**Syntax**

```
CEE3GRN(-name-, -fc-)
```
**name (output)**
A fixed-length 80-character string (VSTRING), that contains the name of the routine that was executing when the condition was raised. name is left-justified within the field and right-padded with blanks. If there are nested conditions, the most recently activated condition is used to determine name.

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE35S</td>
<td>1</td>
<td>3260</td>
<td>No condition was active when a call to a condition management routine was made.</td>
</tr>
</tbody>
</table>

**Usage notes**
- z/OS UNIX consideration—In multithread applications, CEE3GRN gets the name of the routine that incurred the condition on the current thread.

**Examples**
Figure 35 is an example of CEE3GRN called by C/C++.

```c
/*Module/File Name: EDC3GRN */
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>
#ifdef __cplusplus
extern "C" {
#endif
void handler(_FEEDBACK *,_INT4 *,_INT4 *,_FEEDBACK *);
#ifdef __cplusplus
}
#endif
int main(void) {
  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;
  /*
   * register condition handler */
  token = 99;
  routine.address = (_POINTER)&handler;
  routine.nesting = NULL;
  CEEHDLR(&routine,&token,&fc);
  if ( _FBCHECK ( fc , CEE000 ) !=0) {
    printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
    exit (2999);
  }
  /*
   * ...
   */
  printf("%d\n", fc.tok_msgno);
}
```
/* set up any condition sev 2 or higher */
c_1 = 3;
c_2 = 99;
cond_case = 1;
sev = 3;
control = 0;
memcpy(facid,"ZZZ",3);
isi = 0;
CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
facid,&isi,&condtok,&fc);
if ( _FBCHECK ( fc , CEE000 ) !=0 )
  printf("CEENCOD failed with message number %d\n",fc.tok_msgno);
  exit(2999);
}

/* signal condition */
CEESGL(&condtok,&qdata,&fc);
if ( _FBCHECK ( fc , CEE000 ) !=0 )
  printf("CEESGL failed with message number %d\n",fc.tok_msgno);
  exit (2999);
}
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result, _FEEDBACK *newfc) {

  _CHAR80 name;
  _FEEDBACK grnfc;
  
  /* get name of the routine that signal the */
  /* condition */
  CEE3GRN(name,&grnfc);
  if ( _FBCHECK ( grnfc , CEE000 ) !=0 )
    printf("CEESGL failed with message number %d\n",grnfc.tok_msgno);
    exit (2999);
  
  printf("the routine that called this condition");
  printf(" handler is: \n %.80s\n",name);
  *result = 10;
  return;
}

Figure 35. C/C++ Example of CEE3GRN (Part 2 of 2)

Figure 36 on page 161 is an example of CEE3GRN called by COBOL.
IDENTIFICATION DIVISION.
PROGRAM-ID. DRV3GRN.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE PROCEDURE-POINTER.
  01 DENOMINATOR PIC S9(9) BINARY.
  01 NUMERATOR PIC S9(9) BINARY.
  01 RATIO PIC S9(9) BINARY.
  01 TOKEN PIC S9(9) BINARY VALUE 0.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID.
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
REGISTER-HANDLER.
*************************************************
** Register handler
*************************************************
  ** Register handler
  *************************************************
  SET ROUTINE TO ENTRY 'CBL3GRN'.
  CALL 'CEEHDLR' USING ROUTINE, TOKEN, FC.
  IF NOT CEE000 of FC THEN
    DISPLAY 'CEEHDLR failed with msg ' Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
RAISE-CONDITION.
*************************************************
** Cause a zero-divide condition.
*************************************************
  MOVE 0 TO DENOMINATOR.
  MOVE 1 TO NUMERATOR.
  DIVIDE NUMERATOR BY DENOMINATOR
  GIVING RATIO.

UNREGISTER-HANDLER.
*************************************************
** Unregister handler
*************************************************
  CALL 'CEEHDLU' USING ROUTINE, TOKEN, FC.
  IF NOT CEE000 of FC THEN
    DISPLAY 'CEEHDLU failed with msg ' Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
STOP RUN.
END PROGRAM DRV3GRN.

Figure 36. COBOL Example of CEE3GRN (Part 1 of 3)
** CBL3GRN - Call CEE3GRN to get the name of **
** the routine that incurred **
** the condition. **
**
*************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3GRN.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 NAME 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.
  01 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.

Figure 36. COBOL Example of CEE3GRN (Part 2 of 3)
Figure 36. COBOL Example of CEE3GRN (Part 3 of 3)

Figure 37 on page 164 is an example of CEE3GRN called by PL/I.
The CEE3GRO service returns the offset within a failing routine of the most recent condition. If there are nested conditions, the most recently signaled condition is returned.

**Syntax**

```
>>> CEE3GRO(-cond_offset-, -fc-)
```

**cond_offset** *(output)*

An INT4 data type that, upon completion of this service, contains the offset within a failing routine of the most recent condition.

**fc** *(output, optional)*

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:
### CEE3GRO

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE35S</td>
<td>1</td>
<td>3260</td>
<td>No condition was active when a call to a condition management routine was made.</td>
</tr>
</tbody>
</table>

### Examples

Figure 38 is an example of CEE3GRO called by COBOL.

```cobol
CBL LIB,QUOTE,NOOPT
MODULE/Filename: IGZT3GRO
******************************************************************************
** DRV3GRO - Register a condition handler **
** that calls CEE3GRO to determine **
** the offset in the program that **
** incurred the condition. **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRV3GRO.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 ROUTINE PROCEDURE-POINTER.
01 DENOMINATOR PIC S9(9) BINARY.
01 NUMERATOR PIC S9(9) BINARY.
01 RATIO PIC S9(9) BINARY.
01 TOKEN PIC S9(9) BINARY VALUE 0.
01 FC.
02 Condition-Token-Value.
COPY CEEIGCT.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
  REGISTER-HANDLER.
******************************************************************************
** Register handler **
******************************************************************************
SET ROUTINE TO ENTRY "CBL3GRO".
CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
  DISPLAY "CEEHDLR failed with msg "
    Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.

Figure 38. COBOL Example of CEE3GRO (Part 1 of 3)
```
RAISE-CONDITION.
******************************************************************************
** Cause a zero-divide condition
******************************************************************************

MOVE 0 TO DENOMINATOR.
MOVE 1 TO NUMERATOR.
DIVIDE NUMERATOR BY DENOMINATOR
GIVING RATIO.

UNREGISTER-HANDLER.
******************************************************************************
** Unregister handler
******************************************************************************

CALL "CEEHDLU" USING ROUTINE, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEEHDLU failed with msg "
   Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.

STOP RUN.
END PROGRAM DRV3GRO.

******************************************************************************
** **
** CBL3GRO - Call CEE3GRO to get the offset **
** in the routine that incurred **
** the condition. **
** **
******************************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3GRO.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 ROFFSET PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.

Figure 38. COBOL Example of CEE3GRO (Part 2 of 3)
Figure 39 on page 168 is an example of CEE3GRO called by PL/I.
Figure 39. PL/I Example of CEE3GRO (Part 1 of 2)
CEE3INF—Query enclave information

CEE3INF queries and returns to the calling routine the information about the system/subsystem, the environment information, the member languages, and the version of Language Environment associated with this enclave.

Syntax

```plaintext
>>>CEE3INF(-sys/subsys,-env-info,-member-id,-gpid,-fc-)
```

`sys/subsys (input/output)`

As input, `sys/subsys` is considered to be an address to a fullword. As output, the fullword is a 32-bit map that represents the operating system or subsystem on which the enclave is currently running.

- 0  Currently executing in the CICS environment
- 1  Currently executing in a CICS.PIPI environment
- 2-3 Reserved for other specific CICS environments
- 4  Currently executing in a TSO environment
- 5  Currently executing in a Batch environment

Figure 39. PL/I Example of CEE3GRO (Part 2 of 2)
Currently executing in a z/OS UNIX environment
Reserved for future use
Currently executing on z/VSE®
Currently executing on z/OS
Reserved.

env-info (input/output)
As input, env-info is considered to be an address to a fullword. As output, the fullword is a 32-bit map representing the environments that are active in that enclave.
0 Currently executing in a PIPI environment
1 Currently executing in a PIPI-Main environment
2 Currently executing in a PIPI-Sub environment
3 Currently executing in a PIPI-Subdp environment
4 Currently executing in a PICI (Pre-init compatibility interface) environment. For more information, see z/OS XL C/C++ Programming Guide.
5 Currently executing in a nested enclave
6 LRR is active in the current enclave
7 Run-time reuse is active in the current environment
8 XPLINK(ON) is in effect in the current enclave
9 POSIX(ON) RTO in effect in the current enclave
10 At least one pthread has been created in this enclave
11 Currently executing on the IPT
12 Multithreaded fork is in effect in the current enclave
13-14 AMODE Init
  B'00'  AMODE 24
  B'10'  AMODE 31
15 Currently executing in a PIPI-Maindp environment
16-31 Reserved for future use

member-id (input/output)
As input, member-id is considered to be an address to a fullword. As output, the fullword is a 32-bit map representing the member languages that are initialized in that enclave.
0 Reserved
1 Reserved
2 Reserved
3 C/C++
4 Reserved
5 COBOL
6 Reserved
7 Fortran
8-9 Reserved
10 PL/I
11 Enterprise PL/I
12-14 Reserved
15 Reserved
16 Reserved
17-23 Reserved for future use
24-31 Current version number of CEE3INF. This is currently set to 0.

gpid (input/output)
A fullword integer representing the version of Language Environment that created this thread. This fullword can be interpreted as a four-byte hex number as follows:
\[PP|VV|RR|MM\]
**fc (output)**

A feedback code, optional in some languages, that indicates the result of this service. The following feedback codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**Usage notes**

- z/OS UNIX considerations — CEE3INF is allowed only in the thread.
- If you are not interested in any one of the parameters being passed, a 0 must be placed in the slot of those parameters. This indicates that no information is needed regarding that parameter.
- When the PIPI or nested enclave environment bit is on, a subsystem bit may not be set.

**Examples**

*Figure 40* is an example of CEE3INF called by C/C++.

```c
/*Module/File Name: EDC3INF */
/*********************************************************************/
/**/ /* THIS EXAMPLE CALLS CEE3INF TO GET INFORMATION OF THE CURRENT */
/**/ /* ENCLAVE, LIKE THE SYSTEM/SUBSYSTEM, THE ENVIRONMENT INFORMATION, */
/**/ /* THE MEMBER LANGUAGES USED AND THE VERSION OF LANGUAGE ENVIRONMENT.*/
/*********************************************************************/

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

int main(void) {
    _INT4 sys_subsys,env_info,member_id,gpid;
    _FEEDBACK fc;

    /* Calling CEE3INF to get the information */
    CEE3INF(&sys_subsys,&env_info,&member_id,&gpid,fc);

    if ( _FBCHECK(fc,CEE000) !=0 ) {
        printf("CEE3INF failed with message number %d\n", fc.tok_msgno);
    }
    printf("System/Subsystem in hex %08x \n",sys_subsys);
    printf("Environment info in hex %08x \n",env_info);
    printf("Member languages in hex %08x \n",member_id);
    printf("GPID information in hex %08x \n",gpid);
    printf("\n");
}
```

*Figure 40. C/C++ Example of CEE3INF*

*Figure 41 on page 172* is an example of CEE3INF called by COBOL.
Figure 41. COBOL Example of CEE3INF (Part 1 of 2)
Figure 41. COBOL Example of CEE3INF (Part 2 of 2)

Figure 42 is an example of CEE3INF called by PL/I.

```cobol
/* PROCESS MACRO; 
  /*******************************************************************************/
  /* */
  /* Licensed Materials - Property of IBM */
  /* */
  /* 5694-A01 (C) Copyright IBM Corp. 2005 */
  /* All rights reserved */
  /* */
  /* US Government Users Restricted Rights - Use, */
  /* duplication or disclosure restricted by GSA */
  /* ADP Schedule Contract with IBM Corp. */
  /* */
  /*******************************************************************************/
  /* Module/File Name: IBM3INF */
  /*******************************************************************************/
  /** */
  /** Function: CEE3INF - query enclave information */
  /** */
  /** This example calls CEE3INF to gather data about the current */
  /** enclave like the system/subsystem, environment information, */
  /** member languages, and Language Environment version number. */
  /** */
  /*******************************************************************************/
PLI3INF: PROCEDURE OPTIONS (MAIN) REORDER;
  %INCLUDE CEEIBMCT;
DCL SYSPRINT File Output Stream;
DCL SYSSUB REAL FIXED BINARY(31,0),
  ENV_INF REAL FIXED BINARY(31,0),
  MEM_ID REAL FIXED BINARY(31,0),
  GPID REAL FIXED BINARY(31,0);
  /*******************************************************************/
  /* Declare a Language Environment Feedback token. */
  /* 12 Total bytes of storage. */
  /*******************************************************************/
Declare 01 LE_Feedback_Code,
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
  05 Case BIT(2),
  05 Severity BIT(3),
  05 Control BIT(3),
  03 FacID CHAR(3),
  03 ISI REAL FIXED BINARY(31,0);
```

Figure 42. PL/I Example of CEE3INF (Part 1 of 2)
CEE3LNG—Set national language

CEE3LNG sets the current national language. You can also use CEE3LNG to query the current national language. Changing the national language changes the languages in which error messages are displayed and printed, the names of the days of the week, and the names of the months. The current national language setting, as well as previous national language settings that have not been discarded, are recorded on the stack on a LIFO (last in, first out) basis. The current national language setting is always on the top of the stack.

There are two methods of changing the default national language setting with CEE3LNG:

- Specify the new national language setting and place it on top of the stack using a function value of 1 (SET). This discards the previous default setting.
- Specify the new national language setting and place it on top of the stack using a function value of 3 (PUSH). This pushes the previous national language setting down on the stack so that you can later retrieve it by discarding the current setting.

To illustrate the second method, suppose you live in the United States and the code for the United States is specified as the default at installation. If you want to use the French defaults for a certain application, you can use CEE3LNG to PUSH France as the national language setting; then when you want the defaults for the United States, you can POP France from the top of the stack, making the United States the national language setting.

If you specify a desired_language not available on your system, the IBM-supplied default desired_language is used. In Language Environment, this is ENU (mixed-case U.S. English). CEEUOPT, CEEROPT 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 ENU (mixed-case U.S. English) is used.

You can also use the NATLANG run-time option to set the national language.
CEE3LNG affects only the Language Environment NLS and date and time services, not the Language Environment locale callable services or C locale-sensitive functions.

Syntax

```
CEE3LNG(function, desired_language, fc)
```

*function* (input)

A fullword binary integer that specifies the service to perform. The possible values for *function* are:

1—SET

Establishes the *desired_language* specified in the call to CEE3LNG as the current language. In effect, it replaces the current language on the top of the stack with the *desired_language* that you specify. When setting the national language, the *desired_language* is folded to uppercase. "enu" and "ENU", for example, are considered to be the same national language.

2—QUERY

Identifies the current language on the top of the stack to the calling routine by returning it in the *desired_language* parameter of CEE3LNG. The *desired_language* retained as the result of the QUERY function is in uppercase.

3—PUSH

Pushes the *desired_language* specified in the call to CEE3LNG on to the top of the language stack, making it the current language. Previous languages are retained on the stack on a LIFO basis, making it possible to return to a prior language at a later time.

4—POP

Pops the current language off the stack. The previous language that was pushed on to the stack now becomes the new current language. Upon return to the caller, the *desired_language* parameter contains the discarded language. If the stack contains only one language and would be empty after the call, no action is taken and a feedback code indicating such is returned.

*desired_language* (input/output)

A 3-character fixed-length string. The string is not case-sensitive and is used in the following ways for different *functions*:

<table>
<thead>
<tr>
<th>If <em>function</em> is:</th>
<th>Then <em>desired_language</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 or 3</td>
<td>Contains the desired national language identification.</td>
</tr>
<tr>
<td>2</td>
<td>Returns the current language on top of the stack.</td>
</tr>
<tr>
<td>4</td>
<td>Returns the discarded national language.</td>
</tr>
</tbody>
</table>

Note: Language Environment supports only these national languages:

- **ENU** Mixed-case U.S. English
- **UEN** Uppercase U.S. English
- **JPN** Japanese

---

CEE3LNG

Chapter 4. Language Environment callable services

175
fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3BQ</td>
<td>2</td>
<td>3450</td>
<td>Only one language was on the stack when a POP request was made to CEE3LNG. The current language was returned.</td>
</tr>
<tr>
<td>CEE3BR</td>
<td>3</td>
<td>3451</td>
<td>The desired language desired-language for the PUSH or SET function for CEE3LNG was invalid. No operation was performed.</td>
</tr>
<tr>
<td>CEE3BS</td>
<td>3</td>
<td>3452</td>
<td>The function function specified for CEE3LNG was not recognized. No operation was performed.</td>
</tr>
</tbody>
</table>

Usage notes
- PL/I MTF consideration—The CEE3LNG callable service is not supported in PL/I multitasking applications.
- PL/I consideration—When running PL/I with POSIX(ON), CEE3LNG is not supported.
- z/OS UNIX consideration
  - CEE3LNG applies to the enclave. Each enclave has a single current national language setting.
  - Language Environment provides z/OS UNIX-compliant locales as part of the C/C++ run-time library. The locales establish the cultural conventions for locale-sensitive functions in this run-time. The CEESETL callable service and other locale callable services depend on the loaded locale. To change the locale, you can use the setlocale() C/C++ library function or the CEESETL callable service. Locale values do not affect the settings used by the CEE3LNG callable service. For a complete list of locales, see z/OS Language Environment Programming Guide.
  - The CEE3LNG callable services, such as date and time formatting, sorting, and currency symbols are independent of the locale. CEE3LNG affects only Language Environment NLS and date and time services. Mixing CEE3LNG services and locale callable services might produce inconsistent results.

For more information
- See “NATLANG” on page 55 for more information about the NATLANG run-time option.
- For more information about the CEESETL callable service, see “CEESETL—Set locale operating environment” on page 448.
- For more information on setlocale(), see z/OS XL C/C++ Programming Guide.

Examples
Figure 43 on page 177 is an example of CEE3LNG called by C/C++. 
/*Module/File Name: EDC3LNG */

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

int main(void) {
    _FEEDBACK fc;
    _INT4 function;
    _CHAR3 lang;

    // Query the current language setting */
    function = 2; /* function 2 is query */
    CEE3LNG(&function,lang,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {  
        printf("CEE3LNG failed with message number \%d\n", fc.tok_msgno);
        exit(2999);
    }

    /* if the current language is not mixed-case */
    /* American English set the current language to */
    /* mixed-case American English */
    if (memcmp(lang,"ENU",3) != 0) {
        memcpy(lang,"ENU",3);
        function = 1; /* function 1 is set */
        CEE3LNG(&function,lang,&fc);
        if ( _FBCHECK ( fc , CEE000 ) != 0 ) {  
            printf("CEE3LNG failed with message number \%d\n", fc.tok_msgno);
            exit(2999);
        }
    }
}

Figure 43. C/C++ Example of CEE3LNG

Figure 44 is an example of CEE3LNG called by COBOL.

CBL LIB,QUOTE

**************************************************************************
** CBL3LNG - Set national language
** In this example, CEE3LNG is called to query
** the current national language setting. If
** the setting is not mixed-case U.S. English,
** CEE3LNG is called to change the setting.
**
**************************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3LNG.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 FUNCTN PIC S9(9) BINARY.
  01 LANG PIC X(3).
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No  PIC S9(4) BINARY.

Figure 44. COBOL Example of CEE3LNG (Part 1 of 2)
Figure 44. COBOL Example of CEE3LNG (Part 2 of 2)

**Figure 45 on page 179** is an example of CEE3LNG called by PL/I.
/* Module/File Name: IBM3LNG */
****************************************************/
/** */
/** Function: CEE3LNG - set national language */
/** */
/** In this example, CEE3LNG is called to query the */
/** current national language setting. If the */
/** setting is not mixed case American English, */
/** CEE3LNG is called to change the setting to that */
/** */
/**********************************************************/

PLI3LNG: PROC OPTIONS(MAIN);

%INCLUDE CEEIIBM6;
%INCLUDE CEEIIBMCT;

DCL FUNCTN REAL FIXED BINARY(31,0);
DCL LANG CHARACTER(3);
DCL 01 FC /* Feedback token */
03 MsgSev REAL FIXED BINARY(15,0),
03 MsgNo REAL FIXED BINARY(15,0),
03 Flags,
05 Case BIT(2),
05 Severity BIT(3),
05 Control BIT(3),
03 FacID CHAR(3), /* Facility ID */
03 ISI /* Instance-Specific Information */
REAL FIXED BINARY(31,0);

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

/* Call CEE3LNG with function code 2 to query */
/* national language */

CALL CEE3LNG ( FUNCTN, LANG, FC );
IF FBCHECK( FC, CEE000) THEN DO;
  PUT SKIP LIST('The current national language ' ||
|| 'is ' || LANG );
END;
ELSE DO;
  DISPLAY('CEE3LNG failed with msg ' || FC.MsgNo);
  STOP;
END;

/* If the current language is not mixed-case */
/* American English, set it to mixed-case */
/* American English */

IF LANG != 'ENU' THEN DO;
  FUNCTN = 1;
  CALL CEE3LNG ( FUNCTN, 'ENU', FC );
  IF ~ FBCHECK( FC, CEE000) THEN DO;
    DISPLAY('CEE3LNG failed with msg ' || FC.MsgNo );
    STOP;
  END;
END;

Figure 45. PL/I Example of CEE3LNG (Part 1 of 2)
Table 23 lists the supported national language codes.

### Table 23. National Language Codes

<table>
<thead>
<tr>
<th>ID</th>
<th>National Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR</td>
<td>Afrikaans</td>
</tr>
<tr>
<td>ARA</td>
<td>Arab countries</td>
</tr>
<tr>
<td>BGR</td>
<td>Bulgarian</td>
</tr>
<tr>
<td>CAT</td>
<td>Catalan</td>
</tr>
<tr>
<td>CHT</td>
<td>Traditional Chinese</td>
</tr>
<tr>
<td>CHS</td>
<td>Simplified Chinese</td>
</tr>
<tr>
<td>CSY</td>
<td>Czech</td>
</tr>
<tr>
<td>DAN</td>
<td>Danish</td>
</tr>
<tr>
<td>DEU</td>
<td>German</td>
</tr>
<tr>
<td>DES</td>
<td>Swiss German</td>
</tr>
<tr>
<td>ELL</td>
<td>Greek</td>
</tr>
<tr>
<td>ENG</td>
<td>U.K. English</td>
</tr>
<tr>
<td>ENU</td>
<td>U.S. English</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish</td>
</tr>
<tr>
<td>FIN</td>
<td>Finnish</td>
</tr>
<tr>
<td>FRA</td>
<td>French</td>
</tr>
<tr>
<td>FRB</td>
<td>Belgian French</td>
</tr>
<tr>
<td>FRC</td>
<td>Canadian French</td>
</tr>
<tr>
<td>FRS</td>
<td>Swiss French</td>
</tr>
<tr>
<td>HEB</td>
<td>Hebrew</td>
</tr>
<tr>
<td>HUN</td>
<td>Hungarian</td>
</tr>
<tr>
<td>ISL</td>
<td>Icelandic</td>
</tr>
<tr>
<td>ITA</td>
<td>Italian</td>
</tr>
<tr>
<td>ITS</td>
<td>Swiss Italian</td>
</tr>
<tr>
<td>JPN</td>
<td>Japanese</td>
</tr>
<tr>
<td>KOR</td>
<td>Korean</td>
</tr>
<tr>
<td>NLD</td>
<td>Dutch</td>
</tr>
<tr>
<td>NLB</td>
<td>Belgian Dutch</td>
</tr>
</tbody>
</table>
Table 23. National Language Codes (continued)

<table>
<thead>
<tr>
<th>ID</th>
<th>National Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOR</td>
<td>Norwegian - Bokmal</td>
</tr>
<tr>
<td>NON</td>
<td>Norwegian - Nynorsk</td>
</tr>
<tr>
<td>PLK</td>
<td>Polish</td>
</tr>
<tr>
<td>PTG</td>
<td>Portuguese</td>
</tr>
<tr>
<td>PTB</td>
<td>Brazilian Portuguese</td>
</tr>
<tr>
<td>RMS</td>
<td>Rhaeto-Romanic</td>
</tr>
<tr>
<td>ROM</td>
<td>Romanian</td>
</tr>
<tr>
<td>RUS</td>
<td>Russian</td>
</tr>
<tr>
<td>SHC</td>
<td>Serbo-Croatian (Cyrillic)</td>
</tr>
<tr>
<td>SHL</td>
<td>Serbo-Croatian (Latin)</td>
</tr>
<tr>
<td>SKY</td>
<td>Slovakian</td>
</tr>
<tr>
<td>SQI</td>
<td>Albanian</td>
</tr>
<tr>
<td>SVE</td>
<td>Swedish</td>
</tr>
<tr>
<td>THA</td>
<td>Thai</td>
</tr>
<tr>
<td>TRK</td>
<td>Turkish</td>
</tr>
<tr>
<td>UEN</td>
<td>U.S. uppercase English</td>
</tr>
<tr>
<td>URD</td>
<td>Urdu</td>
</tr>
</tbody>
</table>

CEE3MCS—Get default currency symbol

CEE3MCS returns the default currency symbol for the country you specify with `country_code`. For a list of the default settings for a specified country, see Table 32 on page 515.

Syntax

```
CEE3MCS(country_code, currency_symbol, fc)
```

`country_code` (input)

A 2-character fixed-length string representing one of the country codes found in Table 32 on page 515. `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 CEE3CTY callable service, is used.

`currency_symbol` (output)

A 4-character fixed-length string returned to the calling routine. It contains the default currency symbol for the country specified. The currency symbol is left-justified and padded on the right with blanks, if necessary.

`fc` (output)

A feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>
Usage notes

- If you specify a `country_code` that is not valid, the default currency symbol is X'9F404040'. In the United States, it is shown as a '$' followed by three blanks.
- z/OS UNIX considerations—CEE3MCS applies to the enclave. Every enclave has a single current country setting that has a single currency symbol. Every thread in every enclave has the same default.
- Euro considerations—For countries in the European Union that have adopted the Euro as the legal tender, the currency symbol is represented as a hexadecimal string in the default country settings. The value is the code point for the Euro sign, taken from a typical code page for the given country. Of course, the actual graphical representation depends on the code page in use. See Table 32 on page 515 for a list of country settings.

Language Environment supports the Euro as the default currency symbol in the following countries: Austria, Belgium, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, the Netherlands, Portugal, and Spain. As more countries pass the economic and monetary union convergence criteria and adopt the Euro as the legal currency, the default currency symbol will replace the national currency symbol with the Euro sign.

For more information

- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 132 for an explanation of the CEE3CTY callable service.
- See "CEE3MC2—Get default and international currency symbols" on page 185 for an explanation of the CEE3MC2 callable service.

Examples

Figure 46 on page 183 is an example of CEE3MCS called by C/C++.
/*Module/File Name: EDC3MCS */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

    _FEEDBACK fc;
    _CHAR2 country;
    _CHAR4 currency;

    /* get the default currency symbol for Canada */
    memcpy(country,"CA",2);
    CEE3MCS(country, currency, &fc);
    if ( _FBCHECK ( fc , CEE000 ) !=0 ) {
        printf("CEE3MCS failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }
    printf("The default currency symbol for Canada is:" \
        " %.2s\n", currency);
}

Figure 46. C/C++ Example of CEE3MCS

Figure 47 on page 184 is an example of CEE3MCS called by COBOL.
CBL LIB,QUOTE
*Module/File Name: IGZT3MCS
*****************************************************************************
** **
** CBL3MCS - Call CEE3MCS to obtain the **
** default currency symbol **
** **
*****************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3MCS.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 COUNTRY   PIC X(2).
  01 CURSYM    PIC X(4).
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity  PIC S9(4) BINARY.
        04 Msg-No    PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl  PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info    PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBL3MCS.
*****************************************************************************
** Specify country code for the US in the call **
** to CEE3MCS **
*****************************************************************************
MOVE "US" TO COUNTRY.
CALL "CEE3MCS" USING COUNTRY, CURSYM, FC.
*****************************************************************************
** If CEE3MCS runs successfully, display result. **
*****************************************************************************
  IF CEE000 of FC THEN
    DISPLAY "The default currency symbol "
    "for the " COUNTRY " is: " CURSYM
  ELSE
    DISPLAY "CEE3MCS failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
GOBACK.

Figure 47. COBOL Example of CEE3MCS

Figure 48 on page 185 is an example of CEE3MCS called by PL/I.
CEE3MC2

CEE3MC2—Get default and international currency symbols

CEE3MC2 returns the default currency symbol and international currency symbol for the country you specify with country_code. For a list of the default settings for a specified country, see Table 32 on page 515.

**Syntax**

```plaintext
>>>CEE3MC2(-country_code-, -currency_symbol-) 
>>>International_currency_symbol-, -fc-)
```

**country_code (input)**

A 2-character fixed-length string representing one of the country codes found in Table 32 on page 515. 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 CEE3CTY callable service, is used.

**currency_symbol (output)**

A 4-character fixed-length string returned to the calling routine. It contains the
CEE3MC2

default currency symbol for the country specified. The currency symbol is left-justified and padded on the right with blanks, if necessary.

*international_currency_symbol (output)*

A 3-character alphabetic fixed-length string returned to the calling routine. It contains the international currency symbol for the country specified.

*fc (output)*

A feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3C2</td>
<td>2</td>
<td>3458</td>
<td>The country code <em>country_code</em> was not valid for CEE3MC2. The default currency symbol <em>currency_symbol</em> was returned. no international currency symbol was returned.</td>
</tr>
</tbody>
</table>

**Usage notes**

- If you specify a *country_code* that is not valid, the default currency symbol is X'9F404040'. In the United States, it is shown as a '$' followed by three blanks.
- z/OS UNIX considerations—CEE3MC2 applies to the enclave. Every enclave has a single current country setting that has a single currency symbol. Every thread in every enclave has the same default.
- Euro considerations—For countries in the European Union that have adopted the Euro as the legal tender, the currency symbol is represented as a hexadecimal string in the default country settings. The value is the code point for the Euro sign, taken from a typical code page for the given country. Of course, the actual graphical representation depends on the code page in use. See Table 32 on page 515 for a list of country settings.

Language Environment supports the Euro as the default currency symbol in the following countries: Austria, Belgium, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, the Netherlands, Portugal, and Spain. As more countries pass the economic and monetary union convergence criteria and adopt the Euro as the legal currency, the default currency symbol will replace the national currency symbol with the Euro sign.

**For more information**

- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 132 for an explanation of the CEE3CTY callable service.
- See "CEE3MCS—Get default currency symbol" on page 181 for an explanation of the CEE3MCS callable service.

**Examples**

Figure 49 on page 187 is an example of CEE3MC2 called by C/C++. 
/*Module/File Name: EDC3MC2 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _CHAR2 country;
  _CHAR4 currency;
  _CHAR3 international_currency;

  /* get the default currency symbol for Canada */
  memcpy(country,"CA",2);
  CEE3MC2(country,currency,international_currency,&fc);

  if (_FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEE3MC2 failed with message number %d\n",
           fc.tok_msgno);
    exit(2999);
  }

  printf("The default currency symbol for Canada is:
         " %.2s\n",currency);
  printf("The international symbol for Canada is:
         " %.3s\n",international_currency);
}

Figure 49. C/C++ Example of CEE3MC2

Figure 50 on page 188 is an example of CEE3MC2 called by COBOL.
Figure 50. COBOL Example of CEE3MC2

Figure 51 on page 189 is an example of CEE3MC2 called by PL/I.
CEE3MDS—Get default decimal separator

CEE3MDS returns the default decimal separator for the country specified by \textit{country\_code}. For a list of the default settings for a specified country, see Table 32 on page 515.

**Syntax**

\[\text{CEE3MDS} \rightarrow \text{country\_code} \rightarrow \text{decimal\_separator} \rightarrow \text{fc} \rightarrow \]

\textit{country\_code} (input)

A 2-character fixed-length string representing one of the country codes found in Table 32 on page 515. \textit{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 CEE3CTY callable service, is used.
decimal_separator (output)
A 2-character fixed-length string containing the default decimal separator for the country specified. The decimal separator is left-justified and padded on the right with a blank.

fc (output)
A 12-byte feedback code, optional in some languages, indicating the service result. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3C4</td>
<td>2</td>
<td>3460</td>
<td>The decimal separator 'decimal-separator' was truncated and was not defined in CEE3MDS.</td>
</tr>
<tr>
<td>CEE3C5</td>
<td>2</td>
<td>3461</td>
<td>The country code country-code was invalid for CEE3MDS. The default decimal separator 'decimal-separator' was returned.</td>
</tr>
</tbody>
</table>

Usage notes
- If you specify a country_code that is not valid, the default decimal separator is a period (.).
- z/OS UNIX considerations—CEE3MDS applies to the enclave. Every enclave has a single current country setting that has a single decimal separator. Every thread in every enclave has the same default.

For more information
- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 132 for an explanation of the CEE3CTY callable service.

Examples
Figure 52 is an example of CEE3MDS called by C/C++.

```c
/*Module/File Name: EDC3MDS */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc;
    _CHAR2 country,decimal;

    /* get the default decimal separator for Canada */
    memcpy(country,"CA",2);
    CEE3MDS(country,decimal,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEE3MDS failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }
    /* print out the default decimal separator */
    printf("The default decimal separator for");
    printf(" Canada is: %.2s\n",decimal);
}
```

Figure 52. C/C++ Example of CEE3MDS
Figure 53 is an example of CEE3MDS called by COBOL.

CBL LIB,QUOTE
*Module/File Name: IGZT3MDS
*****************************************************************************
** **
** CALL CEE3MDS to get the **
** default decimal separator **
** **
*****************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3MDS.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 COUNTRY PIC X(2).
01 DECSEP PIC X(2).
01 FC.
  02 Condition-Token-Value.
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
*****************************************************************************
** Specify the country code for the US in the **
** call to CEE3MDS. **
** If call was successful, print result. **
*****************************************************************************
PARA-CBL3MDS.
  MOVE "US" TO COUNTRY.
  CALL "CEE3MDS" USING COUNTRY, DECSEP, FC.
  IF CEE000 of FC THEN
    DISPLAY "The default Decimal Separator "
    "for " COUNTRY " is " DECSEP ""
  ELSE
    DISPLAY "CEE3MDS failed with msg "
    "Msg-No of FC upon CONSOLE"
    STOP RUN
  END-IF.
GOBACK.

Figure 53. COBOL Example of CEE3MDS

Figure 54 on page 192 is an example of CEE3MDS called by PL/I.
CEE3MTS—Get Default thousands separator

CEE3MTS returns the default thousands separator for the specified country with
\textit{country\_code}.

**Syntax**

\begin{verbatim}
>>>CEE3MTS(--country\_code--, --thousands\_separator--, \textit{fc})
\end{verbatim}

\textit{country\_code} (input)

A 2-character fixed-length string representing one of the country codes found
in Table 32 on page 515. \textit{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 CEE3CTY callable service, is used.

\textit{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.
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3C8</td>
<td>2</td>
<td>3464</td>
<td>The thousands separator ‘thousands-separator’ was truncated and was not defined in CEE3MTS.</td>
</tr>
<tr>
<td>CEE3C9</td>
<td>2</td>
<td>3465</td>
<td>The country code country-code was invalid for CEE3MTS. The default thousands separator ‘thousands-separator’ was returned.</td>
</tr>
</tbody>
</table>

**Usage notes**

- If you specify a *country_code* that is not valid, the default thousands separator is a comma (,).
- *z/OS UNIX considerations—CEE3MTS applies to the enclave. Every enclave has a single current country setting that has a single thousands separator. Every thread in every enclave has the same default.*

**For more information**

- For a list of the default settings for a specified country, see [Table 32 on page 515](#).
- See [“COUNTRY” on page 22](#) for an explanation of the COUNTRY run-time option.
- See [“CEE3CTY—Set default country” on page 132](#) for an explanation of the CEE3CTY callable service.

**Examples**

**Figure 55** is an example of CEE3MTS called by C/C++.

---

```c
/*Module/File Name: EDC3MTS */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc;
    _CHAR2 country,thousand;

    /* get the default thousands separator for Canada */
    memcpy(country,"CA",2);
    CEE3MTS(country,thousand,&fc);
    if (_FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEE3MTS failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }
    /* print out the default thousands separator */
    printf("The default thousands separator for Canada is: %.2s\n",thousand);
}
```

**Figure 55. C/C++ Example of CEE3MTS**

**Figure 56 on page 194** is an example of CEE3MTS called by COBOL.
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3MTS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 COUNTRY PIC X(2).
01 THOUSEP PIC X(2).
01 FC.
  02 Condition-Token-Value.
COPY CEEIGZC1.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBL3MTS.
*************************************************
** Specify the country code for the US in the**
** call to CEE3MTS.**
*************************************************
MOVE "US" TO COUNTRY.
CALL "CEE3MTS" USING COUNTRY, THOUSEP, FC.

*************************************************
** If CEE3MTS runs successfully, display result**
*************************************************
IF CEE000 of FC THEN
  DISPLAY "The default Thousands Separator"
  " for " COUNTRY " is " THOUSEP 
ELSE
  DISPLAY "CEE3MTS failed with msg "
  Msg-No of FC UPON CONSOLE
END-IF.
GOBACK.

Figure 56. COBOL Example of CEE3MTS
Figure 57 is an example of CEE3MTS called by PL/I.

```pli
*PROCESS MACRO;
/ * Module/File Name: IBM3MTS */
/ *********************************************/
/ ** Function: CEE3MTS - obtain default thousands **/
/ ** separator **/
/ *********************************************/
PLI3MTS: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL COUNTRY CHARACTER (2) ;
DCL THOUSEP CHARACTER (2) ;
DCL 01 FC, /* Feedback token */
     03 MsgSev REAL FIXED BINARY(15,0),
     03 MsgNo REAL FIXED BINARY(15,0),
     03 Flags,
     05 Case BIT(2),
     05 Severity BIT(3),
     05 Control BIT(3),
     03 FacID CHAR(3), /* Facility ID */
     03 ISI /* Instance-Specific Information */
          REAL FIXED BINARY(31,0);

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

/* Call CEE3MTS to return default thousands */
/* separator for the US */
CALL CEE3MTS ( COUNTRY, THOUSEP, FC );

/* If CEE3MTS ran successfully print out result */
IF FBCHECK( FC, CEE000) THEN DO;
   PUT SKIP LIST('The default thousands separator for the ' ||
                  COUNTRY || ' is ' || THOUSEP || '' );
   END;
ELSE DO;
   DISPLAY ('CEE3MTS failed with msg ' ||
             FC.MsgNo );
   STOP;
   END;
END PLI3MTS;
```

**Figure 57. PL/I Example of CEE3MTS**

### CEE3PRM—Query parameter string

CEE3PRM queries and returns to the calling routine the parameter string specified at invocation of the program. The returned parameter string contains only program arguments. If no program arguments are available, a blank string is returned.

**Syntax**

```
>>> CEE3PRM(--char_parm_string--, --fc--)
```

**char_parm_string (output)**

An 80-byte fixed-length string passed by CEE3PRM. On return from this service, the `char_parm_string` contains the parameter string specified at invocation of the program. If this parameter string is longer than 80 characters, it is truncated. If the parameter string is shorter than 80 characters, the returned string is padded with blanks. If the program argument passed to the service is absent, or is not a character string, `char_parm_string` is blank.
**CEE3PRM**

*fc (output)*

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3I1</td>
<td>1</td>
<td>3649</td>
<td>The parameter string returned from CEE3PRM exceeded the maximum length of 80 bytes and was truncated.</td>
</tr>
</tbody>
</table>

**Usage notes**

- **C/C++ consideration**—C/C++ users can use the `__osplist` to return a program argument longer than 80 characters.
- **z/OS UNIX considerations**—CEE3PRM is allowed only in the initial thread.
- You can use the CEE3PR2 callable service to return a program string that is longer than 80 characters.

**For more information**

- For more information about the CEE3PR2 callable service, see "CEE3PR2—Query parameter string long" on page 198

**Examples**

Figure 58 is an example of CEE3PRM called by C/C++.

```c
/*Module/File Name: EDC3PRM */
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceedcct.h>

int main() {
    _CHAR80 parm;
    _FEEDBACK fc;
    CEE3PRM(parm,&fc);
    if (_FBCHECK (fc , CEE000 ) !=0) { 
        printf("CEE3PRM failed with message number %d\n", 
        fc.tok_msgno);
        exit(2999);
    }
    printf("%.80s\n",parm);
}
```

Figure 58. C/C++ Example of CEE3PRM

Figure 59 on page 197 is an example of CEE3PRM called by COBOL.
Figure 59. COBOL Example of CEE3PRM

Figure 60 on page 198 is an example of CEE3PRM called by PL/I.
CEE3PR2—Query parameter string long

CEE3PR2 queries and returns to the calling routine the parameter string and its associated length specified at invocation of the program. The returned parameter string contains only program arguments. If no program arguments are available, a blank string is returned.

Syntax

```
CEE3PR2(---int_parm_length---,---char_parm_string---,---fc---)
```

**int_parm_length (input/output)**

A fullword integer that indicates the length of the parameter string.

- For input: If the value is less than or equal to 0, `int_parm_length` is a request for the length of the parameter string passed during program invocation. If the value is greater than 0, it is the actual size of the `char_parm_string` buffer passed by the caller.
- For output: CEE3PR2 returns the actual length of the parameter string passed during program invocation for any case, regardless of the input value in the `int_parm_length` field.

**char_parm_string (input/output)**
For input: `char_parm_string` contains the address of a piece of storage that is obtained by the caller.

**Note:** This storage is used as a varying string. It must be 2 bytes longer than the length of the parameter string passed during program invocation.

For output: The storage pointed to by this address contains a varying string that contains the parameter string passed during program invocation. The parameter string is truncated when the actual length of the parameter string passed during invocation exceeds `int_parm_length` minus 2.

`fc` *(output)*
A feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3K3</td>
<td>1</td>
<td>3715</td>
<td>The parameter string returned from CEE3PR2 was truncated due to insufficient storage space for the string provided by the caller.</td>
</tr>
</tbody>
</table>

**Usage notes**
- **C/C++ consideration**—C/C++ users can use the `__osplist` return a program argument longer than 80 characters.
- **z/OS UNIX consideration**—CEE3PR2 is allowed only in the initial thread.
- **CICS consideration**—CEE3PR2 always returns a blank string.

**For more information**
- For more information about the CEE3PRM callable service, see "CEE3PRM—Query parameter string" on page 195.

**Examples**
Figure 61 on page 200 is an example of CEE3PR2 called by C/C++. 
Figure 62 on page 201 is an example of CEE3PR2 called by COBOL.
**Figure 63 on page 202** is an example of CEE3PR2 called by PL/I.

---

Figure 62. COBOL Example of CEE3PR2
CEE3PR2

```pli
*PROCESS MACRO;
/channel
/* Licensed Materials - Property of IBM */
/* Copyright IBM Corp. 2005 */
/* All rights reserved */
/* US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA */
/* ADP Schedule Contract with IBM Corp. */
/* */
/channel
/** Module/File Name: IBM3PR2 */
/channel
/** Function: CEE3PR2 - Query Parameter String */
/channel
/** This example calls CEE3PR2 to retrieve the arguments passed during the invocation of the program. */
/channel
/** */
/channel
PLI3PR2: PROCEDURE OPTIONS (MAIN) REORDER;
/channel
%INCLUDE CEEIBMAW;
/channel
%INCLUDE CEEIBMCT;
/channel
DCL SYSPRINT File Output Stream;
/channel
DCL Parm_Len FIXED BINARY(31,0),
[channel]
Parm_Str CHAR(255) VARYING;
/channel
/** Declare a Language Environment Feedback token. */
/channel
/** 12 Total bytes of storage. */
/channel
Declare 01 LE_Feedback_Code,
/channel
03 MsgSev REAL FIXED BINARY(15,0),
03 MsgNo REAL FIXED BINARY(15,0),
03 Flags, 05 Case BIT(2),
05 Severity BIT(3),
05 Control BIT(3),
03 FacID CHAR(3),
03 ISI REAL FIXED BINARY(31,0);
/channel
/** Local declares needed for Messaging Callable Services */
/channel
Declare Msg_String CHAR(255) VARYING;
Declare Output PICTURE '999999';
/channel
Msg_dest = 2;
/channel
Parm_Len = 255;
```

Figure 63. PL/I Example of CEE3PR2 (Part 1 of 2)
CEE3RPH—Set report heading

CEE3RPH sets the heading displayed at the top of the storage or options report generated when you specify the RPTSTG(ON) or RPTOPTS(ON) run-time options. For examples of these reports, see z/OS Language Environment Debugging Guide.

Syntax

```
>>> CEE3RPH(—report_heading— , —fc—)
```

`report_heading (input)`

An 80-character fixed-length string, `report_heading` sets the identifying character string displayed at the top of the storage or options report. Language Environment uses only the first 79 bytes of `report_heading`; the last byte is ignored. `report_heading` can contain DBCS characters surrounded by X'0E' (shift-out) and X'0F' (shift-in).

`fc (output)`

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3JK</td>
<td>0</td>
<td>3700</td>
<td>The storage and options report heading replaced a previous heading.</td>
</tr>
</tbody>
</table>

Usage notes

- PL/I considerations—CEE3RPH is designed to provide an equivalent function to the special PL/I variable PLIXHD.
- z/OS UNIX considerations—CEE3RPH applies to the enclave.

For more information

- See "RPTSTG" on page 68 for more information about the RPTSTG run-time option.
- See "RPTOPTS" on page 66 for more information about the RPTOPTS run-time option.
- See PL/I for MVS & VM Programming Guide for further information on PLIXHD.
Examples

Figure 64 is an example of CEE3RPH called by C/C++.

```c
/*Module/File Name: EDC3RPH */
#pragma runopts(RPTOPTS(ON))
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>
#include <ceedcct.h>

int main(void) {
    _CHAR80 heading;
    /* initialize heading to blanks and then set the */
    /* heading */
    memset(heading,' ',80);
    memcpy(heading,"User Defined Report Heading",27);
    /* set the report heading...do not need to check */
    /* feedback token because all return codes are */
    /* successful */
    CEE3RPH(heading,NULL);
    /* . */
    /* . */
}
```

Figure 64. C/C++ Example of CEE3RPH

Figure 65 on page 205 is an example of CEE3RPH called by COBOL.
** CBL LIB,QUOTE

*Module/File Name: IGZ3RPH
*************************************************
** CBL3RPH - Call CEE3RPH to set report heading**
** **
** In this example, a call is made to CEE3RPH **
** to set the report heading that appears at **
** the top of each page of the options report **
** (generated by RPTOPTS) and storage report **
** (generated by RPTSTG). **
** **
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3RPH.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 RPTHEAD      PIC X(80).
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity  PIC S9(4) BINARY.
      04 Msg-No    PIC S9(4) BINARY.
      03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info     PIC S9(9) BINARY.

PROCEDURE DIVISION.

*************************************************
** Specify user-defined report heading via CEE3RPH
*************************************************
PARA-CBL3RPH.
  MOVE "My options and storage reports heading"
  TO RPTHEAD.
  CALL "CEE3RPH" USING RPTHEAD, FC.
  IF NOT CEE000 of FC THEN
  DISPLAY "CEE3RPH failed with msg "
    Msg-No of FC UPON CONSOLE
  STOP RUN
  END-IF.
  GOBACK.

Figure 65. COBOL Example of CEE3RPH

Figure 66 on page 206 is an example of CEE3RPH called by PL/I.
CEE3SPM—Query and modify Language Environment hardware condition enablement

CEE3SPM queries and modifies the enablement of Language Environment hardware conditions. You can use the CEE3SPM service to:

- Alter the settings of the Language Environment conditions, using an action value of 1 (SET), to those specified by the caller.
- Query the current settings of the Language Environment conditions and return the settings to the caller.
- Push the current settings of the Language Environment conditions on to the condition stack using an action value of 3 (PUSH). This pushes the current setting down on the stack for later retrieval, and, in effect, places a copy of the current setting on top of the stack. It does not alter the current condition settings.
- Pop the pushed settings of the Language Environment conditions from the condition stack using an action value of 4 (POP). This reinstates the previous condition settings as the current condition settings.
- Push the current settings of the Language Environment conditions on to the Language Environment-managed condition stack and alter the settings of the Language Environment conditions to those supplied by the caller.

The enabled or disabled Language Environment conditions are:

*PROCESS MACRO;
/* Module/File Name: IBM3RPH */
/***************************************************************************
* Function: CEE3RPH - set report heading *
***************************************************************************/
**
PLI3RPH: PROC OPTIONS(MAIN);

INCLUDE CEEIBMNW;
INCLUDE CEEIBMCT;

DCL RPTHEAD CHAR(80);
DCL 01 FC,
   / Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
   05 Case BIT(2),
   05 Severity BIT(3),
   05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
   REAL FIXED BINARY(31,0);

/* Define report heading */
RPTHEAD = 'My storage report heading';

/* Set report heading in call to CEE3RPH */
CALL CEE3RPH ( RPTHEAD, FC );

IF FBCHECK( FC, CEE000) THEN DO;
    PUT SKIP LIST('Report heading now set to "' || RPTHEAD || '"');
END;
ELSE DO;
    DISPLAY('CEE3RPH failed with msg ' || FC.MsgNo);
    STOP;
END;

END PLI3RPH;

Figure 66. PL/I Example of CEE3RPH
fixed-overflow
When enabled, raises the fixed-overflow condition when an overflow occurs during signed binary arithmetic or signed left-shift operations.

decimal-overflow
When enabled, raises the decimal-overflow condition when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the results.

underflow
When enabled, raises the underflow condition when the result characteristic of a floating-point operation is less than zero and the result fraction is not zero. For an extended-format floating-point result, the condition is raised only when the high-order characteristic underflows.

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

When you use the CEE3SPM callable service, maintenance of the condition stack is required. For example, one user-written condition handler can disable a hardware condition while another enables it. Therefore, do not assume that the program mask is at a given setting. The program mask is set differently based on different HLL requirements. You can find out what the current setting is by using the QUERY function of CEE3SPM.

Language Environment initialization sets conditions based on the languages in the initial load module. Each language present adds to the conditions that are enabled.

Some S/370 hardware interrupt codes and their matching Language Environment feedback codes appear in Table 24 on page 209.

Syntax

```
CEE3SPM(--action--,cond_string--,fc--)
```

action (input)
The action to be performed. action is specified as a fullword binary signed integer corresponding to one of the numbers in the following list:

1—SET
Alters the settings of the Language Environment conditions to those specified in the cond_string parameter.

2—QUERY
Queries the current settings of the Language Environment conditions and return the settings in the cond_string parameter.

3—PUSH
Pushes the current settings of the Language Environment conditions on to the Language Environment-managed condition stack.

4—POP
Pops the pushed settings of the Language Environment conditions from the condition stack, discarding the current settings, and reinstating the previous condition settings as the current condition settings.

5—PUSH, SET
Pushes the current settings of the Language Environment conditions on to the Language Environment-managed
CEE3SPM

condition stack and alters the settings of the Language Environment conditions to those supplied by the caller in the cond_string parameter.

cond_string (input/output)
A fixed-length string of 80 bytes containing a sequence of identifiers representing the requested settings for the Language Environment conditions that can be enabled and disabled. A list of conditions enabled and disabled and their associated identifiers is given below:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed-overflow</td>
<td>F (NOF for disablement)</td>
</tr>
<tr>
<td>decimal-overflow</td>
<td>D (NOD for disablement)</td>
</tr>
<tr>
<td>underflow</td>
<td>U (NOU for disablement)</td>
</tr>
<tr>
<td>significance</td>
<td>S (NOS for disablement)</td>
</tr>
</tbody>
</table>

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 Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE36V</td>
<td>4</td>
<td>3295</td>
<td>The condition string from CEE3SPM did not contain all of the settings, because the returned string was truncated.</td>
</tr>
<tr>
<td>CEE370</td>
<td>4</td>
<td>3296</td>
<td>Some of the data in the condition string from CEE3SPM could not be recognized.</td>
</tr>
<tr>
<td>CEE371</td>
<td>4</td>
<td>3297</td>
<td>The service completed successfully for recognized condition(s), unsuccessfully for unrecognized (unsupported) condition(s).</td>
</tr>
<tr>
<td>CEE372</td>
<td>4</td>
<td>3298</td>
<td>A call to CEE3SPM attempted to PUSH settings onto a full stack.</td>
</tr>
<tr>
<td>CEE373</td>
<td>4</td>
<td>3299</td>
<td>A call to CEE3SPM attempted to POP settings off an empty stack.</td>
</tr>
<tr>
<td>CEE374</td>
<td>4</td>
<td>3300</td>
<td>The action parameter in CEE3SPM was not one of the digits 1 to 5.</td>
</tr>
</tbody>
</table>

Usage notes

- PL/I MTF consideration—In PL/I MTF applications, CEE3SPM affects only the calling task.
- C/C++ consideration—C/C++ ignores the fixed-overflow, decimal-overflow, underflow, and significance interrupts, no matter what you specify in CEE3SPM.
- COBOL consideration—COBOL ignores the fixed-overflow and decimal-overflow interrupts, no matter what you specify in CEE3SPM.
- z/OS UNIX consideration—In multithread applications, CEE3SPM affects only the calling thread.
- You cannot use CEE3SPM to enable the fixed-overflow, decimal-overflow, underflow or significance interrupts. You can, however, query the settings of CEE3SPM.
Table 24. S/370 Interrupt Code Descriptions

<table>
<thead>
<tr>
<th>S/370 Interrupt Code</th>
<th>Description</th>
<th>Maskable</th>
<th>Symbolic Feedback Code</th>
<th>Message Number</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Operation exception</td>
<td>No</td>
<td>CEE341</td>
<td>3201</td>
<td>3</td>
</tr>
<tr>
<td>0002</td>
<td>Privileged operation exception</td>
<td>No</td>
<td>CEE342</td>
<td>3202</td>
<td>3</td>
</tr>
<tr>
<td>0003</td>
<td>Execute exception</td>
<td>No</td>
<td>CEE343</td>
<td>3203</td>
<td>3</td>
</tr>
<tr>
<td>0004</td>
<td>Protection exception</td>
<td>No</td>
<td>CEE344</td>
<td>3204</td>
<td>3</td>
</tr>
<tr>
<td>0005</td>
<td>Addressing exception</td>
<td>No</td>
<td>CEE345</td>
<td>3205</td>
<td>3</td>
</tr>
<tr>
<td>0006</td>
<td>Specification exception</td>
<td>No</td>
<td>CEE346</td>
<td>3206</td>
<td>3</td>
</tr>
<tr>
<td>0007</td>
<td>Data exception</td>
<td>No</td>
<td>CEE347</td>
<td>3207</td>
<td>3</td>
</tr>
<tr>
<td>0008</td>
<td>Fixed-point overflow exception</td>
<td>Yes</td>
<td>CEE348</td>
<td>3208</td>
<td>3</td>
</tr>
<tr>
<td>0009</td>
<td>Fixed-point divide exception</td>
<td>No</td>
<td>CEE349</td>
<td>3209</td>
<td>3</td>
</tr>
<tr>
<td>000A</td>
<td>Decimal-overflow exception</td>
<td>Yes</td>
<td>CEE34A</td>
<td>3210</td>
<td>3</td>
</tr>
<tr>
<td>000B</td>
<td>Decimal divide exception</td>
<td>No</td>
<td>CEE34B</td>
<td>3211</td>
<td>3</td>
</tr>
<tr>
<td>000C</td>
<td>Exponent-overflow exception</td>
<td>No</td>
<td>CEE34C</td>
<td>3212</td>
<td>3</td>
</tr>
<tr>
<td>000D</td>
<td>Exponent-underflow exception</td>
<td>Yes</td>
<td>CEE34D</td>
<td>3213</td>
<td>3</td>
</tr>
<tr>
<td>000E</td>
<td>Significance exception</td>
<td>Yes</td>
<td>CEE34E</td>
<td>3214</td>
<td>3</td>
</tr>
<tr>
<td>nn0F</td>
<td>Floating-point divide exception</td>
<td>No</td>
<td>CEE34F</td>
<td>3215</td>
<td>3</td>
</tr>
</tbody>
</table>

Examples

Figure 67 is an example of CEE3SPM called by C/C++.

```c
//Module/File Name: EDC3SPM
*
/******                        
/* This example queries the enablement of LE/370 */
/* hardware conditions. */
/******                        */
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

    _FEEDBACK fc;
    _INT4 action;
    _CHAR80 cond_string;
    char *cond;

    /* query the current settings */
    action = 2;
    CEE3SPM(&action,cond_string,&fc);

    if (_FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEE3SPM query failed with message \

                %s\n", fc.tok_msgno);
        exit(2999);
    }
}
```

Figure 67. C/C++ Example of CEE3SPM

Figure 68 on page 210 is an example of CEE3SPM called by COBOL.
Figure 68. COBOL Example of CEE3SPM

Figure 69 on page 211 is an example of CEE3SPM called by PL/I.
Figure 69. PL/I Example of CEE3SPM (Part 1 of 2)
CEE3SRC—Set the enclave return code

CEE3SRC sets the user enclave return code. The value set is used in the calculation of the final enclave return code at enclave termination.

Syntax

```plaintext
CEE3SRC(return_code, fc)
```

return_code (input)

An INT4 data type. The enclave return code to be set should be <= 999,999 and >= 0 to be in the Language Environment-preferred range. (The initial value is 0.)

fc (output)

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0HA</td>
<td>1</td>
<td>0554</td>
<td>A value outside the preferred range of 0 through 999,999 was supplied. However, the value was still used as the enclave return code.</td>
</tr>
</tbody>
</table>

Usage note

- z/OS UNIX consideration—CEE3SRC is not supported in multithread applications.
- PL/I MTF consideration—For PL/I multitasking applications, the user return code value set during invocation of CEE3SRC affects the current task only. If a PL/I program causes an enclave to terminate, the value set by CEE3SRC at the associated task level is reflected in the final return code at enclave termination.
- PL/I consideration—When running PL/I with POSIX(ON), CEE3SRC is not supported.
CEE3SRC

For more information

- See [z/OS Language Environment Programming Guide](#) for more information about the CEE3SRC callable service.

Examples

CEE3SRC is used with CEE3GRC; see the examples for CEE3GRC show in “Examples” on page 150.

CEE3SRP—Set resume point

The CEE3SRP service sets the resume point at the next instruction in the calling routine. CEE3SRP works only in conjunction with the CEEMRCE service.

Syntax

```
CEE3SRP (resume_token, fc)
```

`resume_token` **(output)**

An INT4 data type that, upon completion of this service, contains a token that represents a machine state block, which Language Environment allocates from heap storage. Language Environment automatically frees the heap storage for the machine state block when the routine associated with the stack frame to which it points returns to its caller.

`fc` **(output / optional)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE390</td>
<td>3</td>
<td>3360</td>
<td>The stack frame was not found on the call chain.</td>
</tr>
</tbody>
</table>

Usage notes

- An example for Service Label under COBOL follows:

  ```
  SET-RECOVERY-POINT-PARAGRAPH.
  CALL 'CEE3SRP' USING RECOVERY-POINT FC.
  SERVICE LABEL.
  ERROR-PARAGRAPH.
  * code to do post-error processing
  ```

- Use the CEE3SRP service only with the CEEMRCE service from within a user condition handler. The token returned by this service is used as input to the CEEMRCE service.

- Language Environment automatically frees the heap storage for the machine state block when the routine associated with the stack frame to which it points returns to its caller. Attempts to use the machine state block after it is freed result in unpredictable behavior.

- **COBOL considerations**
  - A Service Label compiler directive must be specified immediately after the call to CEE3SRP.
CEE3SRP

- When a resume occurs and control resumes to the next instruction following the call to CEE3SRP, the COBOL RETURN-CODE special register contains an unpredictable value.

Examples
For examples of how to use CEE3SRP in combination with CEEMRCE and CEEHDLR, see “CEEMRCE—Move resume cursor explicit” on page 384.

CEE3USR—Set or query user area fields
CEE3USR sets or queries one of two 4-byte fields known as the user area fields. The user area fields are associated with an enclave and are maintained on an enclave basis. A user area field can be used by vendor or application programs to store a pointer to a global data area or to keep a recursion counter.

Be careful not to confuse the Language Environment user area fields with the PL/I user area. The PL/I user area is a 4-byte field in the PL/I TCA and can be accessed only through assembler language. The PL/I user area continues to be supported for compatibility.

Language Environment initializes both user area fields to X'00000000' during enclave initialization.

Syntax

```
CEE3USR((function_code,field_number,field_value,fc))
```

function_code (input)
A fullword binary integer representing the function performed:

1—SET User area field according to the value specified in field_value.
2—QUERY User area field; return current value in field_value.

field_number (input)
A fullword binary integer indicating the field to set or query. field_number must be specified as either 1 or 2.

field_value (input/output)
A fullword binary integer.

If function_code is specified as 1 (meaning SET user area field), field_value contains the value to be copied to the user area field.

If function_code is specified as 2 (meaning QUERY user area field), the value in the user area field is copied to field_value.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3PS</td>
<td>3</td>
<td>3900</td>
<td>The function code passed to CEE3USR was not 1 or 2.</td>
</tr>
<tr>
<td>CEE3PT</td>
<td>3</td>
<td>3901</td>
<td>The field number passed to CEE3USR was not 1 or 2.</td>
</tr>
</tbody>
</table>
Usage note
• z/OS UNIX consideration—CEE3USR applies to the enclave.

Examples
Figure 70 is an example of CEE3USR called by C/C++.

```c
/*Module/FileName: EDC3USR*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>

typedef struct {
  int value1,value2,value3;
  char slot1_80 ;
} info_struct;

int main (void) {

  _INT4 function_code, field_number, field_value;
  _FEEDBACK fc;
  info_struct *info;

  info = (info_struct *)malloc(sizeof(info_struct));
  /* . */
  /* Set User field 1 to point to info_struct */
  function_code = 1;
  field_number = 1;
  field_value = (int)info;
  CEE3USR(&function_code,&field_number,&field_value,&fc);
  if ( _FBCHECK ( fc , CEE000 ) !=0 ){
    printf("CEE3USR failed with message number %d\n", fc.tok_msgno);
    exit(2999);
  }
  /* . */
  /* get the value of field 2 */
  function_code = 2;
  field_number = 1;
  CEE3USR(&function_code,&field_number,&field_value,&fc);
  if ( _FBCHECK ( fc , CEE000 ) !=0 ){
    printf("CEE3USR failed with message number %d\n", fc.tok_msgno);
    exit(2999);
  }
}

Figure 70. C/C++ Example of CEE3USR
```

Figure 71 on page 216 is an example of CEE3USR called by COBOL.
CBL LIB,QUOTE
*Module/File Name: IGZT3USR
*************************************************
** CBL3USR - Call CEE3USR to set or query user **
** area fields **
** In this example, CEE3USR is called twice: **
** once to set the value of a user area, and **
** once to query it. **
**
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3USR.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 FUNCODE PIC S9(9) BINARY.
  01 FILONO 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 MsgBox 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-3USRSET.
  MOVE 1 TO FUNCODE.
  MOVE 1 TO FILONO.
  MOVE 23 TO INVALUE.
  CALL "CEE3USR" USING FUNCODE, FILONO, INVALUE, FC.
  IF NOT CEE000 of FC THEN
    DISPLAY "CEE3USR failed with msg " MsgBox of FC UPON CONSOLE
    STOP RUN
  END-IF.

Figure 71. COBOL Example of CEE3USR (Part 1 of 2)
Figure 71. COBOL Example of CEE3USR (Part 2 of 2)

Figure 72 is an example of CEE3USR called by PL/I.

```
*PROCESS MACRO;
/* Module/file Name: IBM3USR */
/***************************************************************************/
/** **
/** Function: CEE3USR - set/query user area fields **
/** **
/** In this example, CEE3USR is called twice: once **
/** to set the value of a user area, and once to **
/** query it. */
/***************************************************************************/
PL3USR: PROC OPTIONS(MAIN);
  %INCLUDE CEEIBM3;
  %INCLUDE CEEIBMCT;
  DCL FUNCODE REAL FIXED BINARY(31,0);
  DCL FIELDNO REAL FIXED BINARY(31,0);
  DCL OUTVALUE REAL FIXED BINARY(31,0);
  DCL INVALUE REAL FIXED BINARY(31,0);
  DCL 01 FC, /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
    03 ISI /* Instance-Specific Information */
  REAL FIXED BINARY(31,0);
  FUNCODE = 1; /* Specify 1 for the set function */
  FIELDNO = 1; /* Specify field 1 of two */
  INVALUE = 5; /* Value to put in field 1 */
```

Figure 72. PL/I Example of CEE3USR (Part 1 of 2)
CEECBLDY

CEECBLDY—Convert date to COBOL Integer format

CEECBLDY converts a string representing a date into a COBOL Integer format, which is the number of days since 1 January 1601. This service is similar to CEEDAYS, except that it provides a string in COBOL Integer format, which is compatible with ANSI intrinsic functions. Use CEECBLDY to access the century window of Language Environment and to perform date calculations with COBOL intrinsic functions for programs compiled with the INTDATE(ANSI) compiler option.

Call CEECBLDY only from COBOL programs that use the returned value as input for COBOL intrinsic functions. You should not use the returned value with other Language Environment callable services, nor should you call CEECBLDY from any non-COBOL programs. Unlike CEEDAYS, there is no inverse function for CEECBLDY, because it is only for COBOL users who want to use the Language Environment century window service together with COBOL intrinsic functions for date calculations. The inverse function for CEECBLDY is provided by the DATE-OF-INTEGER and DAY-OF-INTEGER intrinsic functions.

To handle dates earlier than 1601, add 4000 to each year, convert to Integer, calculate, subtract 4000 from the result, and then convert back to character format. By default, 2-digit years lie within the 100-year range starting 80 years prior to the system date. Thus, in 1995, all 2-digit years represent dates between 1915 and 2014, inclusive. You can change this default range by using the CEESCEN callable service.
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 521] for a list of valid picture character
terms that can be specified in input_char_date.

picture_string (input)
A halfword length-prefixed character string (VSTRING), indicating the format
of the date specified in input_char_date. Each character in the picture_string
corresponds to a character in input_char_date. For example, if you specify
MMDDYY as the picture_string, CEECBLDY reads an input_char_date of 060288
as 02 June 1988. If delimiters such as the slash (/) appear in the picture string,
leading zeros can be omitted. For example, the following calls to CEECBLDY
would each assign the same value, 148155 (02 June 1988), to COBINTDATE:

```
MOVE '6/2/88' TO DATEVAL.
MOVE 'MM/DD/YY' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);

MOVE '06/02/88' TO DATEVAL.
MOVE 'MM/DD/YY' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);

MOVE '060288' TO DATEVAL.
MOVE 'MMDDYY' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);

MOVE '88154' TO DATEVAL.
MOVE 'YYDDD' TO PICSTR.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDATE, fc);
```
Whenever characters such as colons or slashes are included in the picture_string
(such as HH:MI:SS YY/MM/DD), they count as placeholders but are otherwise
ignored. See [Table 33 on page 521] for a list of valid picture character terms and
[Table 34 on page 522] 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 522] for an additional example. See also [Table 35 on page 522]
for a list of Japanese Eras supported by CEEDATE.
If picture_string includes era symbol <CCCC> or <CCCCCCCC>, the YY position in input_char_date is replaced by the year number within the era. See Table 34 on page 522 for an additional example.

output_Integer_date (output)
A 32-bit binary integer representing the COBOL Integer date, the number of days since 31 December 1600. For example, 16 May 1988 is day number 141485. If input_char_date does not contain a valid date, output_Integer_date is set to 0 and CEECBLDY terminates with a non-CEE000 symbolic feedback code. Date calculations are performed easily on the output_Integer_date, because output_Integer_date is an integer. Leap year and end-of-year anomalies do not affect the calculations.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2EB</td>
<td>3</td>
<td>2507</td>
<td>Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.</td>
</tr>
<tr>
<td>CEE2EC</td>
<td>3</td>
<td>2508</td>
<td>The date value passed to CEEDAYS or CEESECS was invalid.</td>
</tr>
<tr>
<td>CEE2ED</td>
<td>3</td>
<td>2509</td>
<td>The era passed to CEEDAYS or CEESECS was not recognized.</td>
</tr>
<tr>
<td>CEE2EH</td>
<td>3</td>
<td>2513</td>
<td>The input date passed in a CEEISEC, CEEDAYS, or CEESECS call was not within the supported range.</td>
</tr>
<tr>
<td>CEE2EL</td>
<td>3</td>
<td>2517</td>
<td>The month value in a CEEISEC call was not recognized.</td>
</tr>
<tr>
<td>CEE2EM</td>
<td>3</td>
<td>2518</td>
<td>An invalid picture string was specified in a call to a date/time service.</td>
</tr>
<tr>
<td>CEE2EO</td>
<td>3</td>
<td>2520</td>
<td>CEEDAYS detected non-numeric data in a numeric field, or the date string did not match the picture string.</td>
</tr>
<tr>
<td>CEE2EP</td>
<td>3</td>
<td>2521</td>
<td>The (&lt;JJJJ&gt;) or (&lt;CCCC&gt;) year-within-era value passed to CEEDAYS or CEESECS was zero.</td>
</tr>
</tbody>
</table>

Usage notes
• The probable cause for receiving message number 2518 is a picture string that contains an invalid DBCS string. You should verify that the data in the picture string is correct.
• z/OS UNIX consideration—In multithread applications, CEECBLDY affects only the calling thread.

For more information
• See the INTDATE COBOL compiler installation option in Enterprise COBOL for z/OS, V4R2, Programming Guide, SC23-8529, Enterprise COBOL for z/OS, V3R4 Programming Guide, SC27-1412, or COBOL for OS/390 & VM Programming Guide for information about how to get ANSI integer values from COBOL Intrinsic Functions that are compatible with the Language Environment callable services CEEDAYS and CEEDATE.
• See “CEESCEN—Set the century window” on page 428 for more information about the CEESCEN callable service.
• See Table 33 on page 521 for a list of valid picture character terms that can be specified in input_char_date.
Examples

Figure 73 is an example of CEECBLDY called by COBOL.

```
CBL LIB,QUOTE
*Module/File Name: IGZTCBLD
******************************************************************************
** Function: Invoke CEECBLDY callable service **
** to convert date to COBOL Integer format. **
** This service is used when using the **
** Lang. Environ. Century Window **
** mixed with COBOL Intrinsic Functions. **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDY.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 CHRDATE.
   02 Vstring-length PIC S9(4) BINARY.
   02 Vstring-text.
      03 Vstring-char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of CHRDATE.

01 PICSTR.
   02 Vstring-length PIC S9(4) BINARY.
   02 Vstring-text.
      03 Vstring-char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of PICSTR.

01 INTEGER PIC S9(9) BINARY.
01 NEWDATE PIC 9(8).
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
PARA-CBLDAYS.
******************************************************************************
** Specify input date and length **
******************************************************************************
MOVE 25 TO Vstring-length of CHRDATE.
MOVE "1 January 00" to Vstring-text of CHRDATE.
```

Figure 73. COBOL Example of CEECBLDY (Part 1 of 2)
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

```
 CEECMI(---cond_rep---,---insert_seq_num---,---insert_data---,---fc---)
```

**cond_rep** (input/output)
A condition token that defines the condition for which the q_data_token is retrieved.

**insert_seq_num** (input)
A 4-byte integer that contains the insert sequence number (such as insert 1 insert 2). It corresponds to an insert number specified with an ins tag in the message source file created by the CEEBLDTX EXEC.

**insert_data** (input)
A halfword-prefixed length string that represents the insert data. The entire length described in the halfword prefix is used without truncation. DBCS

Figure 73. COBOL Example of CEECBLDY (Part 2 of 2)
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 Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0EB</td>
<td>3</td>
<td>0459</td>
<td>Not enough storage was available to create a new instance specific information block.</td>
</tr>
<tr>
<td>CEE0EC</td>
<td>1</td>
<td>0460</td>
<td>Multiple instances of the condition token with message number message-number and facility ID facility-id were detected.</td>
</tr>
<tr>
<td>CEE0ED</td>
<td>3</td>
<td>0461</td>
<td>The maximum number of unique message insert blocks was reached. This condition token had its I_S_info field set to 1.</td>
</tr>
<tr>
<td>CEE0EE</td>
<td>3</td>
<td>0462</td>
<td>Instance specific information for the condition token with message number message-number and facility ID facility-id could not be found.</td>
</tr>
<tr>
<td>CEE0EF</td>
<td>3</td>
<td>0463</td>
<td>The maximum size for an insert data item was exceeded.</td>
</tr>
<tr>
<td>CEE0H9</td>
<td>3</td>
<td>0553</td>
<td>An internal error was detected in creating the inserts for a condition.</td>
</tr>
</tbody>
</table>

Usage notes
• z/OS UNIX consideration—In multithread applications, CEECMI applies to message insert data for only the calling thread.

For more information
• See "MSGQ" on page 55 for more information about the MSGQ run-time option.
• For more information about CEEBLDTX, see z/OS Language Environment Programming Guide.

Examples
Figure 74 on page 224 is an example of CEECMI called by C/C++.
/* Module Name: EDCCMI */
/**************************************************
** FUNCTION: CEENCOD - set up a condition token **
** : CEECOMI - store and load message **
** : CEEUSG - retrieve, format, and **
** dispatch a message to **
** : CEEMSG - retrieve, format, and **
** dispatch a message to **
** ***************
** This example illustrates the invocation of **
** the Lang. Environ. message services to **
** store and load message insert data. **
** The resulting message and insert is written **
** to the Lang. Environ. MSGFILE ddname. **
** ***************
**************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

void main ()
{
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;
  _VSTRING insert;
  _FEEDBACK ctok;
  _FEEDBACK fbcode;
  _INT4 MSGFILE;
  _INT4 insert_no ;

  /* Condition Token Declarations */
  /* **********************************
  * EXMPLMSG is a token that represents message *
  * number 10 in a user message file constructed *
  * using the CEEBLDTXT facility. *
  * Message 10 is designed to allow one insert. *
  * **********************************/ 
  insert.length = 18;
  memcpy(insert.string ,"CEPGOMI's insert",
  insert.length);

  /*give ctok value of hex 0000000A40E7D4D70000000 */
  /*sev = 0 msgno = 10 facid = XMP */
  /*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);
}

Figure 74. C/C++ Example of CEECMIO (Part 1 of 2)
Figure 75. COBOL Example of CEECMI (Part 1 of 2)

Figure 75 is an example of CEECMI called by COBOL.
**Figure 76 on page 227** is an example of CEECMI called by PL/I.
CEECRHP—Create new additional heap

CEECRHP lets you define additional heaps. It returns a unique heap_id. The heaps defined by CEECRHP can be used just like the initial heap (heap_id=0), below heap, and anywhere heap. Unlike the heaps created by these heap services, all heap elements within an additional heap can be quickly freed by a single call to CEEDSHP (discard heap). The number of heaps supported by Language Environment is limited only by the amount of virtual storage available.

Figure 76. PL/I Example
heap_id (output)
A fullword binary signed integer. heap_id is the heap identifier of the created heap. If a new heap cannot be created, the value of heap_id remains undefined. Storage obtained from heap_ids 79 and 80 is set to binary 0 independent of any initialization value specified by the STORAGE option.

initial_size (input)
A fullword binary signed integer. initial_size is the initial amount of storage, in bytes, allocated for the new heap. initial_size is rounded up to the nearest increment of 4096 bytes. If initial_size is specified as 0, then the init_size specified in the HEAP 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 25.

Table 25. HEAP Attributes Based on the Setting of the options Parameter

<table>
<thead>
<tr>
<th>Option Setting</th>
<th>HEAP Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Use same attributes as the initial heap (copy them from the HEAP run-time option)</td>
</tr>
<tr>
<td>01</td>
<td>HEAP(,,FREE) (location inherited from HEAP run-time option)</td>
</tr>
<tr>
<td>70</td>
<td>HEAP(,,KEEP) (location inherited from HEAP run-time option)</td>
</tr>
<tr>
<td>71</td>
<td>HEAP(,,ANYWHERE,KEEP)</td>
</tr>
<tr>
<td>72</td>
<td>HEAP(,,ANYWHERE,KEEP)</td>
</tr>
<tr>
<td>73</td>
<td>HEAP(,,BELOW,KEEP)</td>
</tr>
<tr>
<td>74</td>
<td>HEAP(,,BELOW,KEEP)</td>
</tr>
<tr>
<td>75</td>
<td>HEAP(,,ANYWHERE,) (disposition inherited from the HEAP run-time option)</td>
</tr>
<tr>
<td>76</td>
<td>HEAP(,,BELOW,) (disposition inherited from the HEAP run-time option)</td>
</tr>
<tr>
<td>77</td>
<td>HEAP(,,ANYWHERE,KEEP) (all heap storage obtained using this heap_id is allocated on a 4K boundary)</td>
</tr>
<tr>
<td>78</td>
<td>HEAP(,,ANYWHERE,KEEP) (all heap storage obtained using this heap_id is allocated on a 4K boundary)</td>
</tr>
</tbody>
</table>
Table 25. HEAP Attributes Based on the Setting of the options Parameter (continued)

<table>
<thead>
<tr>
<th>Option Setting</th>
<th>HEAP Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>HEAP(,,ANYWHERE,KEEP) (all heap storage obtained using this heap_id is set to binary 0 when allocated using CEEGTST)</td>
</tr>
<tr>
<td>80</td>
<td>HEAP(,,ANYWHERE,FREE) (all heap storage obtained using this heap_id is set to binary 0 when allocated using CEEGTST)</td>
</tr>
</tbody>
</table>

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0P2</td>
<td>4</td>
<td>0802</td>
<td>Heap storage control information was damaged.</td>
</tr>
<tr>
<td>CEE0P4</td>
<td>3</td>
<td>0804</td>
<td>The initial size value supplied in a create heap request was unsupported.</td>
</tr>
<tr>
<td>CEE0P5</td>
<td>3</td>
<td>0805</td>
<td>The increment size value supplied in a create heap request was unsupported.</td>
</tr>
<tr>
<td>CEE0P6</td>
<td>3</td>
<td>0806</td>
<td>The options value supplied in a create heap request was unrecognized.</td>
</tr>
<tr>
<td>CEE0PD</td>
<td>3</td>
<td>0813</td>
<td>Insufficient storage was available to satisfy a get storage request.</td>
</tr>
</tbody>
</table>

Usage notes
- z/OS UNIX consideration—CEECRHP applies to the enclave.
- The heapid can only be used by the TCB on which the CEECRHP request was issued. Using the heapid on other TCBs is not supported and will generate unpredictable results.

For more information
- See “CEEDSHP—Discard heap” on page 268 for more information about the CEEDSHP callable service.
- See “HEAP” on page 36 for more information about the HEAP run-time option and IBM-supplied defaults.
- See “HEAP” on page 36 for more information about the HEAP run-time option and IBM-supplied defaults.

Examples
Figure 77 on page 230 is an example of CEECRHP called by C/C++. 
Figure 77. C/C++ Example of CEECRHP

Figure 78 on page 231 is an example of CEECRHP called by COBOL.
**Figure 79 on page 232** is an example of CEECRHP called by PL/I.
CEECZST—Reallocate (change size of) storage

CEECZST changes the size of a previously allocated heap element. The address parameter points to the beginning of the heap element. The new size parameter gives the new size of the heap element, in bytes. The contents of the heap element are unchanged up to the shorter of the new and old sizes.

The CEECZST service returns a pointer to the reallocated heap element. It can move the storage location of the heap element. As a result, the address parameter passed to CEECZST is not necessarily the same as the value returned.

Because the new storage might be allocated at a different location from the existing allocation, any pointers (specifically any addresses) that referred to the old storage become invalid. Continued use of such dangling pointers gives unpredictable, and almost certainly incorrect, results.

Figure 79. PL/I Example of CEECRHP
The heap identifier is inferred from the address. The new storage block is allocated from the same heap that contained the old block.

The contents of the old storage are preserved in the following manner:
- If `new_size` is greater than the old size, the entire contents of the old storage block are copied to the new block. The remaining bytes in the new element are left uninitialized unless an initialization suboption value was specified for the heap in the STORAGE option.
- If `new_size` is less than the old size, the contents of the old block are truncated to the size of the new block.
- If `new_size` is equal to the old size, no operations are performed; a successful feedback code is returned.

Syntax

```
>>—CEECZST—(—address—,—new_size—,—fc—)
```

**address (input/output)**
A fullword address pointer. On input, this parameter contains an address returned by a previous CEEGTST call. On output, the address of the first byte of the newly allocated storage is returned in this parameter.

**new_size (input)**
A fullword binary signed integer. `new_size` is the number of bytes of storage to be allocated for the new heap element.

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0P2</td>
<td>4</td>
<td>0802</td>
<td>Heap storage control information was damaged.</td>
</tr>
<tr>
<td>CEE0P8</td>
<td>3</td>
<td>0808</td>
<td>Storage size in a get storage request or a re-allocate request was not a positive number.</td>
</tr>
<tr>
<td>CEE0PA</td>
<td>3</td>
<td>0810</td>
<td>The storage address in a free storage request was not recognized, or heap storage control information was damaged.</td>
</tr>
<tr>
<td>CEE0PD</td>
<td>3</td>
<td>0813</td>
<td>Insufficient storage was available to satisfy a get storage request.</td>
</tr>
</tbody>
</table>

**Usage notes**
- Storage that is reallocated maintains the same mark/release status as the old storage block. If the old storage block was marked, the new storage block carries the same mark and is released by a release operation that specifies that mark.
- z/OS UNIX consideration—CEECZST applies to the enclave.
- The _CEE_REALLOC_CONTROL environment variable provides additional levels of storage control, which can aid CEECZST to more efficiently use storage. See [_CEE_REALLOC_CONTROL| in [z/OS XL C/C++ Programming Guide](http://www.ibm.com) for more information about the environment variable.
**CEECZST**

**For more information**
- See "STORAGE" on page 75 for more information about the STORAGE run-time option.
- For information about CEEGTST, see “CEEGTST—Get heap storage” on page 331.

**Examples**

Figure 80 is an example of CEECZST called by C/C++.

```c
/* Module/File Name: EDCCZST */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>
int main(void) {
  _INT4 heapid, size;
  _POINTER address;
  _FEEDBACK fc;
  /* . . . */
  heapid = 0; /* get storage from initial heap */
  size = 4000; /* number of bytes of heap storage */
  /* obtain the storage using CEEGTST */
  CEEGTST(&heapid,&size,&address,&fc);
  /* check the first 4 bytes of the feedback token */
  /* (0 if successful) */
  if ( _FBCHECK ( fc , CEE000 ) !=0 ) {
    printf("CEEGTST failed with message number %d\n", fc.tok_msgno);
    exit(99);
  }
  /* . . . */
  size = 2000; /* new size of storage element */
  /* change the size of the storage element */
  CEECZST(&address,&size,&fc);
  /* check the first 4 bytes of the feedback token */
  /* (0 if successful) */
  if ( _FBCHECK ( fc , CEE000 ) !=0 ) {
    printf("CEECZST failed with message number %d\n", fc.tok_msgno);
    exit(99);
  }
  /* . . . */
  /* free the storage that was previously obtained */
  /* using CEEGTST */
  CEEFRST(&address,&fc);
  /* check the first 4 bytes of the feedback token */
  /* (0 if successful) */
  if ( _FBCHECK ( fc , CEE000 ) !=0 ) {
    printf("CEEFRST failed with message number %d\n", fc.tok_msgno);
    exit(99);
  }
  /* . . . */
  }
```

Figure 80. C/C++ Example of CEECZST

Figure 81 on page 235 is an example of CEECZST called by COBOL.
CBL LIB,QUOTE
*Module/File Name: IGZTCST
************************************************
** Function: CEECZST - reallocate storage **
** **
** In this example, CEEGTST is called to **
** request storage from HEAPID = 0, and **
** CEECZST is called to change the size of **
** that storage request. **
** **
** ************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLCZST.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HEAPID PIC S9(9) BINARY.
01 HPSIZE PIC S9(9) BINARY.
01 ADDRSS POINTER.
01 NEWSIZE PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLGTST.
*************************************************
** Specify 0 to get storage from the initial **
** heap. Specify 4000 to get 4000 bytes of **
** storage. **
*************************************************
MOVE 0 TO HEAPID.
MOVE 4000 TO HPSIZE.
*************************************************
** Call CEEGTST to obtain storage. **
*************************************************
CALL "CEEGTST" USING HEAPID, HPSIZE, ADDRSS, FC.
*************************************************
** If CEEGTST runs successfully, display result**
*************************************************
IF CEE000 OF FC THEN
   DISPLAY " " HPSIZE
   " bytes have been allocated."
ELSE
   DISPLAY "CEEGTST failed with msg "
   Msg-No of FC UPON CONSOLE
   STOP RUN
END-IF.
Figure 81. COBOL Example of CEECZST (Part 1 of 2)
Figure 81. COBOL Example of CEECZST (Part 2 of 2)

Figure 82 is an example of CEECZST called by PL/I.

```cobol
*PROCESS MACRO;
/*Module/File Name: IBMGZST */
/***********************************************************/
/** */
/** 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 CEEIIMAW;
%INCLUDE CEEIOMCT;

DCL HEAPID REAL FIXED BINARY(31,0) ;
DCL STGSIZE REAL FIXED BINARY(31,0) ;
DCL ADDRSS1 POINTER;
DCL 01 FC1, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
   05 Case BIT(2),
   05 Severity BIT(3),
   05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);
```

Figure 82. PL/I Example of CEECZST (Part 1 of 2)
CEEDATE—Convert Lilian date to character format

CEEDATE converts a number representing a Lilian date to a date written in character format. The output is a character string, such as 1993/09/09.

Do not use CEEDATE in combination with COBOL intrinsic functions.

The inverse of CEEDATE is CEEDAYS, which converts character dates to the Lilian format.

CEEDATE is affected only by the country code setting of the COUNTRY run-time option or CEE3CTY callable service, not the CEESETL callable service or the setlocale() function.
CEEDATE

Syntax

```plaintext
CEEDATE(input_Lilian_date, picture_string, output_char_date, fc)
```

**input_Lilian_date (input)**

A 32-bit integer representing the Lilian date. The Lilian date is the number of days since 14 October 1582. For example, 16 May 1988 is Lilian day number 148138. The valid range of Lilian dates is 1 to 3,074,324 (15 October 1582 to 31 December 9999).

**picture_string (input)**

A halfword length-prefixed character string (VSTRING), representing the desired format of `output_char_date`, for example `MM/DD/YY`. Each character in `picture_string` represents a character in `output_char_date`. If delimiters such as the slash (/) appear in the picture string, they are copied to `output_char_date`.

See Table 33 on page 521 for a list of valid picture characters, and Table 34 on page 522 for examples of valid picture strings.

If `picture_string` is null or blank, CEEDATE gets `picture_string` based on the current value of the COUNTRY 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 `MM/DD/YY HH:MM:SS AM (or PM)`, for example: 09/09/93 4:56:29 PM. This default mechanism makes it easy for translation centers to specify the preferred date, and for applications and library routines to use this format automatically.

If `picture_string` includes a Japanese Era symbol `<JJJJ>`, the `YY` position in `output_char_date` is replaced by the year number within the Japanese Era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 34 on page 522 for an additional example. Also see Table 35 on page 522 for a list of Japanese Eras supported by CEEDATE.

If `picture_string` includes a era symbol `<CCCC>` or `<CCCCCCCC>`, the `YY` position in `output_char_date` is replaced by the year number within the era. See Table 34 on page 522 for an example.

**output_char_date (output)**

A fixed-length 80-character string (VSTRING), is the result of converting `input_Lilian_date` to the format specified by `picture_string`. See Table 26 on page 243 for sample output dates. If `input_Lilian_date` is not valid, `output_char_date` is set to all blanks. CEEDATE terminates with a non-CEE000 symbolic feedback code.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2EG</td>
<td>3</td>
<td>2512</td>
<td>The Lilian date value was not within the supported range.</td>
</tr>
<tr>
<td>CEE2EM</td>
<td>3</td>
<td>2518</td>
<td>An incorrect picture string was specified.</td>
</tr>
<tr>
<td>Code</td>
<td>Severity</td>
<td>Message number</td>
<td>Message text</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CEE2EQ</td>
<td>3</td>
<td>2522</td>
<td><code>&lt;JJJJ&gt;</code>, <code>&lt;CCCC&gt;</code> or <code>&lt;CCCCCCCC&gt;</code> 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.</td>
</tr>
<tr>
<td>CEE2EU</td>
<td>2</td>
<td>2526</td>
<td>The date string returned by CEEDATE was truncated.</td>
</tr>
</tbody>
</table>

**Usage notes**

- The probable cause for receiving message number 2518 is a picture string that contains a DBCS string that is not valid. You should verify that the data in the picture string is correct.
- To create a null VSTRING, set the length to zero; the content of the text portion does not matter. To create a blank VSTRING, any length greater than zero can be used; the content of the text portion must be spaces or blanks.
- **z/OS UNIX consideration**—In multithread applications, CEEDATE applies to the enclave.

**For more information**

- See the INTDATE COBOL compiler installation option in [Enterprise COBOL for z/OS, V4R2, Programming Guide](SC23-8529), [Enterprise COBOL for z/OS, V3R4 Programming Guide](SC27-1412), or [COBOL for OS/390 & VM Programming Guide](SC27-1412) for information about how to get Lilian integer values from COBOL intrinsic functions that are compatible with the Language Environment callable services CEEDAYS and CEEDATE.
- See "CEEDAYS—Convert date to Lilian format" on page 250 for more information about the CEEDAYS callable service.
- See "COUNTRY" on page 22 for more information about the COUNTRY run-time option.
- See "CEEFMDA—Get default date format" on page 284 for information about how to get the default format for a given country code.

**Examples**

Figure 83 on page 240 is an example of CEEDATE called by C/C++.
Figure 83. C/C++ Example of CEEDATE

```c
/*Module/File Name: EDCDATE */
#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceedccct.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 Wwwwmmmzz, Mmmmmmmmmzz 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
",fc.tok_msgno);
        exit(2999);
    }
    printf("%.80s
",date_out);
}
```

Figure 84 is an example of CEEDATE called by COBOL.

Figure 84. COBOL Example of CEEDATE (Part 1 of 3)
Figure 84. COBOL Example of CEEDATE (Part 2 of 3)

CEEDATE

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 Mmmmmmmmmmmmmm 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.
**Figure 84. COBOL Example of CEEDATE (Part 3 of 3)**

*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 CEEIMMW;
%INCLUDE CEEIMMCT;
DCL LILIAN REAL FIXED BINARY(31,0) ;
DCL PICSTR CHAR(255) VARYING;
DCL CHRDATE CHAR(80) ;
DCL 01 FC, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,  
  05 Case BIT(2),
  05 Severity BIT(3),
  05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI * Instance-Specific Information */
                REAL FIXED BINARY(31,0);
LILIAN = 152385; /* input date in Lilian format */
/* picture string that describes how converted */
/* date is to be formatted */
PICSTR = 'ZD Mmmmmmz 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 LISTE 'Lilian day ' || LILIAN  
          ' is equivalent to ' || CHRDATE ;
END;
ELSE DO;
  DISPLAY 'CEEDATE failed with msg ' 
        || FC.MsgNo ;
  STOP;
END;
END PLIDATE;

**Figure 85. PL/I Example of CEEDATE**
Table 26 shows the sample output from CEEDATE.

Table 26. Sample Output of CEEDATE

<table>
<thead>
<tr>
<th>input_Lilian_date</th>
<th>picture_string</th>
<th>output_char_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>148138</td>
<td>YY</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>YYMM</td>
<td>8805</td>
</tr>
<tr>
<td></td>
<td>YY-MM</td>
<td>88-05</td>
</tr>
<tr>
<td></td>
<td>YYMMDD</td>
<td>880516</td>
</tr>
<tr>
<td></td>
<td>YYYY-MM-DD</td>
<td>19880516</td>
</tr>
<tr>
<td></td>
<td>YYYY-ZM-ZD</td>
<td>1988-5-16</td>
</tr>
<tr>
<td></td>
<td>JJJJ YY.MM.DD</td>
<td>Showa 63.05.16</td>
</tr>
<tr>
<td></td>
<td>(in a DBCS string)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCCC YY.MM.DD</td>
<td>Min Guo 77.05.16</td>
</tr>
<tr>
<td></td>
<td>(in a DBCS string)</td>
<td></td>
</tr>
<tr>
<td>148139</td>
<td>MM</td>
<td>05</td>
</tr>
<tr>
<td></td>
<td>MMDD</td>
<td>0517</td>
</tr>
<tr>
<td></td>
<td>MM/DD</td>
<td>05/17</td>
</tr>
<tr>
<td></td>
<td>MMDDYY</td>
<td>051788</td>
</tr>
<tr>
<td></td>
<td>MM/DD/YYYY</td>
<td>05/17/1988</td>
</tr>
<tr>
<td></td>
<td>ZM/DD/YYYY</td>
<td>5/17/1988</td>
</tr>
<tr>
<td>148140</td>
<td>DD</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>DDDM</td>
<td>1805</td>
</tr>
<tr>
<td></td>
<td>DDDMMYY</td>
<td>180588</td>
</tr>
<tr>
<td></td>
<td>DD.MM,YY</td>
<td>18.05.88</td>
</tr>
<tr>
<td></td>
<td>DD.MM,YYYY</td>
<td>18.05.1988</td>
</tr>
<tr>
<td></td>
<td>DD Mmm YYYY</td>
<td>18 May 1988</td>
</tr>
<tr>
<td>148141</td>
<td>DDD</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>YYDD</td>
<td>88148</td>
</tr>
<tr>
<td></td>
<td>YY.DDD</td>
<td>88.140</td>
</tr>
<tr>
<td></td>
<td>YYYY.DDD</td>
<td>1988.140</td>
</tr>
<tr>
<td>148142</td>
<td>YY/MM/DD HH:MI:SS.99</td>
<td>88/05/20</td>
</tr>
<tr>
<td></td>
<td>YYYY/ZM/ZD ZH:MI AP</td>
<td>00:00:00.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1988/5/20 0:00 AM</td>
</tr>
<tr>
<td>148143</td>
<td><a href="http://WWW">WWW</a>., MMM DD, YYYY</td>
<td>SAT., MAY 21,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1988</td>
</tr>
<tr>
<td></td>
<td><a href="http://WWW">WWW</a>., Mmm DD, YYYY</td>
<td>Sat., May 21,</td>
</tr>
<tr>
<td></td>
<td>Wwwwwwwwwww Mmmmmmmmm</td>
<td>1988</td>
</tr>
<tr>
<td></td>
<td>DD, YYYY</td>
<td>Saturdaybb, Mayyyyyy 21,</td>
</tr>
<tr>
<td></td>
<td>wwwwwwwwwz, Mmmmmmmmmz</td>
<td>1988</td>
</tr>
<tr>
<td></td>
<td>DD, YYYY</td>
<td>Saturday, May 21,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1988</td>
</tr>
</tbody>
</table>

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 CEE3CTY callable service, not the CEESETL callable service or the setlocale() function.
CEEDATM

Syntax

```
CEEDATM(input_seconds, picture_string, output_timestamp, fc)
```

**input_seconds (input)**

A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range of input_seconds is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999).

**picture_string (input)**

A halfword length-prefixed character string (VSTRING), representing the desired format of output_timestamp, for example, MM/DD/YY HH:MI AP.

Each character in the picture_string represents a character in output_timestamp. If delimiters such as a slash (/) appear in the picture string, they are copied as is to output_timestamp.

See Table 33 on page 521 for a list of valid picture character terms and Table 34 on page 522 for examples of valid picture strings.

If picture_string is null or blank, CEEDATM gets picture_string based on the current value of the COUNTRY 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”.

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 522 for an example. See Table 35 on page 522 for a list of Japanese Eras supported by CEEDATM.

If picture_string includes era symbol <CCCC> or <CCCCCCCC>, the YY position in output_timestamp represents the year within the era. See Table 34 on page 522 for an example.

**output_timestamp (output)**

A fixed-length 80-character string (VSTRING), that is the result of converting input_seconds to the format specified by picture_string. If necessary, the output is truncated to the length of output_timestamp. See Table 27 on page 249 for sample output.

If input_seconds is not valid, output_timestamp is set to all blanks and CEEDATM terminates with a non-CEE000 symbolic feedback code.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2E9</td>
<td>3</td>
<td>2505</td>
<td>The number-of-seconds value was not within the supported range.</td>
</tr>
</tbody>
</table>
### Usage notes

- The probable cause for receiving message number 2518 is a picture string that contains a DBCS string that is not valid. You should verify that the data in the picture string is correct.
- To create a null VSTRING, set the length to zero; the content of the text portion does not matter. To create a blank VSTRING, any length greater than zero can be used; the content of the text portion must be spaces or blanks.
- z/OS UNIX consideration—In multithread applications, CEEDATM applies to the enclave.

### For more information

- See “CEESECS—Convert timestamp to seconds” on page 441 for more information about the CEESECS callable service.
- See “COUNTRY” on page 22 for more information about the COUNTRY run-time option.
- See “CEEFMDT—Get default date and time format” on page 287 for information about how to get a default timestamp for a given country code.

### Examples

Figure 86 on page 246 is an example of CEEDATM called by C/C++. 

### CEEDATM

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE2EA</td>
<td>3</td>
<td>2506</td>
<td>&lt;JJJJ&gt;, &lt;CCCC&gt; or &lt;CCCCCCCC&gt; 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.</td>
</tr>
<tr>
<td>CEE2EM</td>
<td>3</td>
<td>2518</td>
<td>An invalid picture string was specified.</td>
</tr>
<tr>
<td>CEE2EV</td>
<td>2</td>
<td>2527</td>
<td>The timestamp string returned by CEEDATM was truncated.</td>
</tr>
<tr>
<td>CEE3CF</td>
<td>2</td>
<td>3471</td>
<td>The country code country-code was not valid for CEEFMDT. The default date and time picture string datetime-string was returned.</td>
</tr>
</tbody>
</table>
CEEDATM

```c
/*Module/File Name: EDCDATM */
#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
    /* September 13, 1991 at 11:23:23 PM */
    _FLOAT8 seconds = 12904183403.0;
    _VSTRING date_pic;
    _CHAR80 out_date;
    _FEEDBACK fc;
    strcpy(date_pic.string, "Mmmmmmmmmmmzz DD, YYYY at ZZ: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
",
               fc.tok_msgno);
        exit(2999);
    }
    printf("%.80s
",out_date);
}
```

**Figure 86. C/C++ Example of CEEDATM**

**Figure 87.** is an example of CEEDATM called by COBOL.

**CBL LIB,QUOTE**

```cobol
*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.
```

**Figure 87. COBOL Example of CEEDATM (Part 1 of 3)**
CEEDATM

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 MMMM yyyy at HH:MI:SS" TO Vstring-text OF PICSTR.

Figure 87. COBOL Example of CEEDATM (Part 2 of 3)
CALL "CEEDATM" USING SECONDS, PICSTR, TIMESTP, FC.

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
END-IF.
GOBACK.

Figure 87. COBOL Example of CEEDATM (Part 3 of 3)

Figure 88 on page 249 is an example of CEEDATM called by PL/I.
Table 27 shows the sample output of CEEDATM.

Table 27. Sample Output of CEEDATM

<table>
<thead>
<tr>
<th>input_seconds</th>
<th>picture_string</th>
<th>output_timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>12,799,191,601.000</td>
<td>YYMMDD</td>
<td>880516</td>
</tr>
<tr>
<td></td>
<td>HH:MM:SS</td>
<td>19:00:01</td>
</tr>
<tr>
<td></td>
<td>YY-MM-DD</td>
<td>88-05-16</td>
</tr>
<tr>
<td></td>
<td>YYMMDHMMISS</td>
<td>880516190001</td>
</tr>
<tr>
<td></td>
<td>YY-MM-DD HH:MM:SS</td>
<td>88-05-16</td>
</tr>
<tr>
<td></td>
<td>YYYY-MM-DD HH:MM:SS</td>
<td>19:00:01</td>
</tr>
<tr>
<td></td>
<td>AP</td>
<td>1988-05-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>07:00:01 PM</td>
</tr>
</tbody>
</table>
### Table 27. Sample Output of CEEDATM (continued)

<table>
<thead>
<tr>
<th>input_seconds</th>
<th>picture_string</th>
<th>output_timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>12,799,191,661.986</td>
<td>DD Mmm YY</td>
<td>16 May 88</td>
</tr>
<tr>
<td></td>
<td>DD MMM YY HH:MM</td>
<td>16 MAY 88 19:01</td>
</tr>
<tr>
<td></td>
<td>WWW, MMM DD, YYYY</td>
<td>MON, MAY 16,</td>
</tr>
<tr>
<td></td>
<td>ZH:MI AP</td>
<td>1988 7:01 PM</td>
</tr>
<tr>
<td></td>
<td>Wwwwwwwwwwz, ZM/ZD/YY</td>
<td>Monday, 5/16/88</td>
</tr>
<tr>
<td></td>
<td>HH:MI:SS.99</td>
<td>19:01:01.98</td>
</tr>
<tr>
<td>12,799,191,662.009</td>
<td>YYYY</td>
<td>1988</td>
</tr>
<tr>
<td></td>
<td>YY</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>MM</td>
<td>05</td>
</tr>
<tr>
<td></td>
<td>ZM</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>RRRR</td>
<td>Vbbb</td>
</tr>
<tr>
<td></td>
<td>MMM</td>
<td>MAY</td>
</tr>
<tr>
<td></td>
<td>Mmm</td>
<td>Maybbbb</td>
</tr>
<tr>
<td></td>
<td>Mmmmmmmmm</td>
<td>May</td>
</tr>
<tr>
<td></td>
<td>Mmmmmmmmmmmz</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>DD</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>ZD</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>DDD</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>HH</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>ZH</td>
<td>02</td>
</tr>
<tr>
<td></td>
<td>MI</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>SS</td>
<td>009</td>
</tr>
<tr>
<td></td>
<td>99</td>
<td>PM</td>
</tr>
<tr>
<td></td>
<td>999</td>
<td>MON</td>
</tr>
<tr>
<td></td>
<td>AP</td>
<td>Mon</td>
</tr>
<tr>
<td></td>
<td>WW</td>
<td>Mondaybbbbb</td>
</tr>
<tr>
<td></td>
<td>Ww</td>
<td>Monday</td>
</tr>
<tr>
<td></td>
<td>Wwwwwwwwww</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wwwwwwwwz</td>
<td></td>
</tr>
</tbody>
</table>

### CEEDAYS—Convert date to Lilian format

CEEDAYS converts a string representing a date into a Lilian format, which represents a date as the number of days from the beginning of the Gregorian calendar. CEEDAYS converts the specified `input_char_date` to a number representing the number of days since day one in the Lilian format: Friday, 14 October, 1582.

The inverse of CEEDAYS is CEEDATE, which converts `output_Lilian_date` from Lilian format to character format.

Do not use CEEDAYS in combination with COBOL intrinsic functions unless the programs are compiled with the INTDATE(LILIAN) compiler option. Use CEECBLDY for COBOL programs that use intrinsic functions and that are compiled with INTDATE(ANSI).

To handle dates earlier than 1601, it is possible to add 4000 to each year, convert to Lilian, calculate, subtract 4000 from the result, and then convert back to character format.

By default, 2-digit years lie within the 100-year range starting 80 years prior to the system date. Thus, in 1995, all 2-digit years represent dates between 1915 and 2014, inclusive. This default range is changed by using the callable service CEESCEN.
CEEDAYS

Syntax

```
CEEDAYS(input_char_date, picture_string, output_Lilian_date, fc)
```

**input_char_date (input)**

A halfword length-prefixed character string (VSTRING), representing a date or timestamp, in a format conforming to that specified by `picture_string`.

The character string must contain between 5 and 255 characters, inclusive. `input_char_date` can contain leading or trailing blanks. Parsing for a date begins with the first nonblank character (unless the `picture_string` itself contains leading blanks, in which case CEEDAYS skips exactly that many positions before parsing begins).

After parsing a valid date, as determined by the format of the date specified in `picture_string`, CEEDAYS ignores all remaining characters. Valid dates range between and include 15 October 1582 to 31 December 9999. See Table 33 on page 521 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 MM/DD/YY as the `picture_string`, CEEDAYS reads an `input_char_date` of 060288 as 02 June 1988.

If delimiters such as a slash (/) appear in the `picture_string`, leading zeros can be omitted. For example, the following calls to CEEDAYS, would each assign the same value, 148155 (02 June 1988), to `lildate`.

```
CALL CEEDAYS('6/2/88' , 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('06/02/88', 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('060288' , 'MMDDYY' , lildate, fc);
CALL CEEDAYS('88154' , 'YYDDD' , lildate, fc);
CALL CEEDAYS('1988154' , 'YYYYDDD' , lildate, fc);
```

Whenever characters such as colons or slashes are included in the `picture_string` (such as HH:MI:SS YY/MM/DD), they count as placeholders but are otherwise ignored. See Table 33 on page 521 for a list of valid picture character terms and Table 34 on page 522 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 522 for an additional example. See Table 35 on page 522 for a list of Japanese Eras supported by CEEDATE.

If `picture_string` includes era symbol <CCCC> or <CCCCCCCC>, the YY position in `input_char_date` is replaced by the year number within the era. See Table 34 on page 522 for an additional example.
CEEDAYS

output_Lilian_date (output)
A 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582. For example, 16 May 1988 is day number 148138.

If input_char_date does not contain a valid date, output_Lilian_date is set to 0 and CEEDAYS terminates with a non-CEE000 symbolic feedback code.

Date calculations are performed easily on the output_Lilian_date, because it is an integer. Leap year and end-of-year anomalies do not affect the calculations.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2EB</td>
<td>3</td>
<td>2507</td>
<td>Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.</td>
</tr>
<tr>
<td>CEE2EC</td>
<td>3</td>
<td>2508</td>
<td>The date value passed to CEEDAYS or CEESECS was not valid.</td>
</tr>
<tr>
<td>CEE2ED</td>
<td>3</td>
<td>2509</td>
<td>The era passed to CEEDAYS or CEESECS was not recognized.</td>
</tr>
<tr>
<td>CEE2EH</td>
<td>3</td>
<td>2513</td>
<td>The input date was not within the supported range.</td>
</tr>
<tr>
<td>CEE2EL</td>
<td>3</td>
<td>2517</td>
<td>The month value was not recognized.</td>
</tr>
<tr>
<td>CEE2EM</td>
<td>3</td>
<td>2518</td>
<td>An incorrect picture string was specified.</td>
</tr>
<tr>
<td>CEE2EO</td>
<td>3</td>
<td>2520</td>
<td>CEEDAYS detected non-numeric data in a numeric field, or the date string did not match the picture string.</td>
</tr>
<tr>
<td>CEE2EP</td>
<td>3</td>
<td>2521</td>
<td>The year-within-era value passed to CEEDAYS or CEESECS was zero.</td>
</tr>
</tbody>
</table>

Usage notes
• The probable cause for receiving message number 2518 is a picture string that contains a DBCS string that is not valid. You should verify that the data in the picture string is correct.
• z/OS UNIX consideration—In multithread applications, CEEDAYS applies to the enclave.

For more information
• See the INTDATE COBOL compiler installation option in Enterprise COBOL for z/OS, V4R2, Programming Guide SC23-8529, Enterprise COBOL for z/OS, V3R4, Programming Guide SC27-1412, or COBOL for OS/390 & VM Programming Guide for information about how to get Lilian integer values from COBOL intrinsic functions that are compatible with the Language Environment callable services CEEDAYS and CEEDATE.
• See “CEEDATE—Convert Lilian date to character format” on page 237 for more information about the CEEDATE run-time option.
• See “CEESCCEN—Set the century window” on page 428 for more information about the CEESECS callable service.

Examples
Figure 89 on page 253 is an example of CEEDAYS called by C/C++.
Figure 89. C/C++ Example of CEEDAYS

```c
#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcc.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,"Mmmmmmdd, 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);
}
```

Figure 90 on page 254 is an example of CEEDAYS called by COBOL.
CEEDAYS

**Module/File Name: IGZTDDAYS**

**Function: CEEDAYS - convert date to Lilian format**

```cobol
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**

```

Figure 90. COBOL Example of CEEDAYS (Part 1 of 2)
Figure 91 is an example of CEEDAYS called by PL/I.

********************************************
** Specify a picture string that describes **
** input date, and the picture string's length.**
********************************************
MOVE 25 TO Vstring-length of PICSTR.
MOVE "ZD Mmmmmmmmmmmmmm 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.

Figure 90. COBOL Example of CEEDAYS (Part 2 of 2)

Figure 91 is an example of CEEDAYS called by PL/I.

*PROCESS MACRO;
/*Module/File Name: IBMDAYS */
******************************************************************************
/* ** Function : CEEDAYS - Convert date to **
/* ** Lilian format **
/* ** This example converts two dates to the Lilian **
/* ** format in order to calculate the number of **
/* ** days between them. **
/* *********************************************************************************/
PLIDAYS: PROC OPTIONS(MAIN);
   %INCLUDE CEEIBMCT;
   %INCLUDE CEEIDMCT;
   DCL CHRDATE CHAR(255) VARYING;
   DCL CHRDATE2 CHAR(255) VARYING;
   DCL PICSTR CHAR(255) VARYING;
   DCL PICSTR2 CHAR(255) VARYING;
   DCL LILIAN REAL FIXED BINARY(31,0);
   DCL LILIAN2 REAL FIXED BINARY(31,0);
   DCL 01 FC, /* Feedback token */
       03 MsgSev REAL FIXED BINARY(15,0),
       03 MsgNo REAL FIXED BINARY(15,0),
       03 Flags,
       05 Case BIT(2),
       05 Severity BIT(3),
       05 Control BIT(3),
       03 FacID CHAR(3), /* Facility ID */
       03 ISI /* Instance-Specific Information */
       REAL FIXED BINARY(31,0);
   /* First date to be converted to Lilian format */
CEEDCOD

CEEDCOD—Decompose a condition token

CEEDCOD alters an existing condition token. Language Environment-conforming HLLs can decompose or alter the condition token fields without using the CEEDCOD service. See the CEESGL HLL examples in Figure 200 on page 456 for examples of how to alter the condition token field.

Syntax

```
CEEDCOD(—cond_token—,—c_1—,—c_2—,—case—,—severity—,———)

-control—,—facility_ID—,—i_s_info—,—fc—)
```

cond_token (input)

A 12-byte condition token representing the current condition or feedback information.

c_1 (output)

A 2-byte binary integer representing the value of the first 2 bytes of the condition_ID.
**CEEDCOD**

*c_2*(output)
A 2-byte binary integer representing the value of the second 2 bytes of the condition_ID. See "CEENCOD—Construct a condition token" on page 407 for a detailed explanation of the condition_ID.

case(output)
A 2-byte binary integer field defining the format of the condition_ID portion of the token. A value of 1 identifies a case 1 condition. A value of 2 identifies a case 2 condition. The values 0 and 3 are reserved.

severity(output)
A 2-byte binary integer representing the severity of the condition. severity specifies the following values:

- 0: Information only (or, if the entire token is zero, no information).
- 1: Warning—service completed, probably correctly.
- 2: Error detected—correction attempted; service completed, perhaps incorrectly.
- 3: Severe error—service not completed.
- 4: Critical error—service not completed; condition signaled. A critical error is a condition that jeopardizes the environment. If a critical error occurs during a Language Environment callable service, instead of returning synchronously to the caller, the condition manager is always signaled.

control(output)
A 2-byte binary integer containing flags describing aspects of the state of the condition. Valid values for the control field are 1 and 0.

- 1: Indicates that the facility_ID is assigned by IBM.
- 0: indicates the facility_ID is assigned by the user.

facility_ID(output)
A 3-character field containing three alphanumeric characters identifying the product generating the condition or feedback information.

i_s_info
A fullword binary integer that identifies the ISI associated with the given instance of the condition represented by the condition token where it is contained. If an ISI is not associated with a given condition token, the i_s_info field contains a value of binary zero.

fc(output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE036</td>
<td>3</td>
<td>0102</td>
<td>An unrecognized condition token was passed to routine and could not be used.</td>
</tr>
</tbody>
</table>

**Usage notes**

- C/C++ considerations—The structure of the condition token (type_FEEDBACK) is described in the leawi.h header file shipped with Language Environment. C users can assign values directly to the fields of the token in the header file without using the CEENCOD service. The layout of the type_FEEDBACK condition token in the header file is shown in Figure 175 on page 409.
CEEDCOD

- z/OS UNIX consideration—In multithread applications, CEEDCOD affects only the calling thread.

For more information
- See the CEESGL HLL examples starting in Figure 200 on page 456 for examples of how to alter the condition token field.
- See “CEENCOD—Construct a condition token” on page 407 for a detailed explanation of the condition_ID.
- See CEENCOD “Usage notes” on page 409 for a discussion of case 1 and case 2 types.
- See the facility_ID parameter of “CEENCOD—Construct a condition token” on page 407 for more information.
- For more C user information about assigning values directly to the fields of the token in the header file without using the CEENCOD service, see the example for “CEESGL—Signal a condition” on page 453.

Examples
Figure 92 on page 259 shows an example of CEEDCOD being called by C/C++.
Figure 93 on page 260 shows an example of CEEDCOD being called by COBOL.
**Function: CEEDCOD - Decompose a condition token**

In this example, a call is made to CEGTST in order to obtain a condition token to use in the call to CEEDCOD. A call could also have been made to any other Lang Env svc., or a condition token could have been constructed using CEEDCOD.

 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLDCOD.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 HEAPID PIC S9(9) BINARY.
 01 HPSIZE PIC S9(9) BINARY.
 01 ADDR55 USAGE POINTER.
 01 SEV PIC S9(4) BINARY.
 01 MSGNO PIC S9(4) BINARY.
 01 CASE PIC S9(4) BINARY.
 01 SEV2 PIC S9(4) BINARY.
 01 CNTRL PIC S9(4) BINARY.
 01 FACID PIC X(3).
 01 ISINFO PIC S9(9) BINARY.
 01 FC.
   02 Condition-Token-Value.
     COPY CEEIGZCT.
       03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
         03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
         03 Case-Sev-Ctl PIC X.
         03 Facility-ID PIC XXX.
         02 I-S-Info PIC S9(9) BINARY.
 01 FC2.
   02 Condition-Token-Value.
     COPY CEEIGZCT.
       03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
         03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
         03 Case-Sev-Ctl PIC X.
         03 Facility-ID PIC XXX.
         02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
******************************************************************************
** Call any Lang Env svc to receive a condition**
** token to use as input to CEEDCOD. **
******************************************************************************

PARA-CBLGTST.

Figure 93. COBOL Example of CEEDCOD (Part 1 of 2)
Figure 94 is an example of CEEDCOD called by PL/I.

** Specify 0 to get storage from the initial **
** heap. **
** Specify 4000 to get 4000 bytes of storage. **
** Call CEEGTST to obtain storage. **

************************************************
** 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.

Figure 93. COBOL Example of CEEDCOD (Part 2 of 2)

Figure 94 is an example of CEEDCOD called by PL/I.

*PROCESS MACRO;
   /*Module/File Name: IBMDCOD */
   /*******************************************/
   /**
   ** Function: CEEDCOD - decompose a condition **
   ** Token **
   /**
   ** In this example, a call is made to CEEGTST to /**
   ** receive a condition token to decompose. **
   ** A call could have been made to any LE/370 /**
   ** service. The condition token returned by /**
   ** CEEGTST is used as input to CEEDCOD. **
   /**
   **************************************************/
   DCL HEAPID REAL FIXED BINARY(31,0);
   DCL STGSIZE REAL FIXED BINARY(31,0);
   DCL ADDRSS POINTER;
   DCL 01 FC, /* Feedback token */
      03 MsgSev REAL FIXED BINARY(15,0),
      03 MsgNo REAL FIXED BINARY(15,0),
      03 Flgs,
      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);

Figure 94. PL/I Example of CEEDCOD (Part 1 of 2)
CEEDLYM—Suspend processing of the active enclave in milliseconds

CEEDLYM provides a service for Language Environment-conforming applications that suspends the processing of the active enclave for a specified number of milliseconds. The maximum is 1 hour.

**Syntax**

```
CEEDLYM(input_milliseconds, fc)
```

**input_milliseconds**
A full-word binary value in the range of 0 to 3,600,000 that specifies the total number of milliseconds during which the enclave should be suspended.

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

```
DCL SEV REAL FIXED BINARY(15,0);
DCL MSGNO REAL FIXED BINARY(15,0);
DCL CASE REAL FIXED BINARY(15,0);
DCL SEV2 REAL FIXED BINARY(15,0);
DCL CNTRL REAL FIXED BINARY(15,0);
DCL FACID CHARACTER(3);
DCL ISINFO REAL FIXED BINARY(31,0);
DCL 01 FC2, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
  05 Case BIT(2),
  05 Severity BIT(3),
  05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);
HEAPID = -1; /* invalid heap ID */
STGSIZE = 4000; /* request 4000 bytes of storage */

/* Call any service (in this case, CEEGTST) to */
/* create a condition token to decompose */
CALL CEEGTST( HEAPID , STGSIZE , ADDRSS , FC );
/* Call CEEDCOD with the condition token */
/* returned in FC from CEEGTST */
CALL CEEDCOD( FC , SEV , MSGNO , CASE , SEV2 ,
  CNTRL , FACID , ISINFO , FC2 );
IF FBCHECK( FC2 , CEE000 ) THEN DO;
  PUT SKIP LIST( 'Feedback token from CEEGTST has'
    || ' Severity of ' || SEV
    || ', Message Number of ' || MSGNO
    || ', Case of ' || CASE
    || ', 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;
```

Figure 94. PL/I Example of CEEDCOD (Part 2 of 2)
### Usage notes

- CICS consideration—CEEDLYM is not available under CICS.
- z/OS UNIX consideration—CEEDLYM is handled by the z/OS UNIX System Services when POSIX is set to ON.
- This service is not intended for timing requests. Delays up to the nearest millisecond might occur in some circumstances.
- In a multi-threaded application, only the calling thread is suspended.

### Examples

[Figure 95 on page 264](#) is an example of CEEDLYM called by C/C++.
Figure 95. C/C++ Example of CEEDLYM

```c
/* Module/File Name: EDCDLYM */
/* */
/* THIS EXAMPLE CALLS CEEDLYM TO SUSPEND PROCESSING OF THE ACTIVE */
/* ENCLAVE FOR SPECIFIED NUMBER OF MILLISECONDS. */
/* */
/* *************************************************************************/
#include <string.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void)
{
    _INT4 millisecs, lil_date;
    _FLOAT8 local_date,
    _CHAR17 gregorian_date;
    _FEEDBACK fc1, fc2;

    /* Get current date and time */
    CEELOCT(&lil_date,&local_date,gregorian_date,&fc1);
    if ( _FBCHECK ( fc1 , CEE000 ) !=0 ) { 
        printf("CEELOCT failed with message number %d\n", fc1.tok_msgno);
        exit(2999);
    }
    printf("CEEDLYM Start time : %.17s\n", gregorian_date);
    millisecs = 5000;
    CEEDLYM(&millisecs,&fc2);

    if ( _FBCHECK ( fc2 , CEE000 ) != 0 ){
    printf("CEEDLYM failed with message number %d\n",fc2.tok_msgno);
    exit(2999);
    }
    CEELOCT(&lil1_date,&local_date,gregorian_date,&fc1);
    if ( _FBCHECK ( fc1 , CEE000 ) != 0 ) { 
        printf("CEELOCT failed with message number %d\n", fc1.tok_msgno);
        exit(2999);
    }
    printf("CEEDLYM Finish time : %.17s\n",gregorian_date);
}
```

Figure 96 on page 265 is an example of CEEDLYM called by COBOL.
**FUNCTION:** CEEDLYM - SUSPEND ENCLAVE EXECUTION IN MILLISECONDS  
**THIS EXAMPLE CALLS CEEDLYM TO SUSPEND PROCESSING OF THE ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF MILLISECONDS.**

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDLYM.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LILIAN PIC 9(9) BINARY.
01 SECONDS COMP-2.
01 GREGORN PIC X(17).
01 MILLISECS PIC 9(9) BINARY.
01 FC1.
   02 CONDITION-TOKEN-VALUE.
      COPY CEEIGZCT.
      03 CASE-1-CONDITION-ID.
         04 SEVERITY PIC 9(4) BINARY.
         04 MSG-NO PIC 9(4) BINARY.
      03 CASE-2-CONDITION-ID
         REDEFINES CASE-1-CONDITION-ID.
         04 CLASS-CODE PIC 9(4) BINARY.
         04 CAUSE-CODE PIC 9(4) BINARY.
      03 CASE-SEV-CTL PIC X.
      03 FACILITY-ID PIC XXX.
      02 I-S-INFO PIC 9(9) BINARY.
   01 FC2.
      02 CONDITION-TOKEN-VALUE.
      COPY CEEIGZCT.
      03 CASE-1-CONDITION-ID.
         04 SEVERITY PIC 9(4) BINARY.
         04 MSG-NO PIC 9(4) BINARY.
      03 CASE-2-CONDITION-ID
         REDEFINES CASE-1-CONDITION-ID.
         04 CLASS-CODE PIC 9(4) BINARY.
         04 CAUSE-CODE PIC 9(4) BINARY.
      03 CASE-SEV-CTL PIC X.
      03 FACILITY-ID PIC XXX.
      02 I-S-INFO PIC 9(9) BINARY.
PROCEDURE DIVISION.

Figure 96. COBOL Example of CEEDLYM (Part 1 of 2)
CEEDLYM

PARA-CBLLYM.
DISPLAY "CEEDLYM - BEGINS"
CALL "CEELOCT" USING LILIAN, SECONDS, GREGORN, FC1.
IF CEE000 OF FC1 THEN
  DISPLAY "START DATE & TIME: " GREGORN
ELSE
  DISPLAY "CEELOCT FAILED WITH MSG "
  MSG-NO OF FC1
  STOP RUN
END-IF.
MOVE 5000 TO MILLISECS.
CALL "CEEDLYM" USING MILLISECS, FC2.
IF NOT CEE000 OF FC2 THEN
  DISPLAY "CEEDLYM FAILED WITH MSG "
  MSG-NO OF FC2
  STOP RUN
END-IF.
DISPLAY "CEEDLYM - COMPLETED"
CALL "CEELOCT" USING LILIAN, SECONDS,
  GREGORN, FC1.
IF CEE000 OF FC1 THEN
  DISPLAY "FINISH DATE & TIME: " GREGORN
ELSE
  DISPLAY "CEELOCT FAILED WITH MSG "
  MSG-NO OF FC1
  STOP RUN
END-IF.
GOBACK.

Figure 96. COBOL Example of CEEDLYM (Part 2 of 2)

Figure 97 on page 267 is an example of CEEDLYM called by PL/I.
CEEDLYM

```
+PROCESS MACRO;
/*****************************/ 00001000
/* Licensed Materials - Property of IBM */ 00004000
/* 5694-A01 (C) Copyright IBM Corp. 2006 */ 00006000
/* All rights reserved */ 00007000
/* US Government Users Restricted Rights - Use, */ 00008000
/* duplication or disclosure restricted by GSA */ 00009000
/* ADP Schedule Contract with IBM Corp. */ 00010000
/* */ 00012000
/*****************************/ 00013000
*/MODULE/FILE NAME: IBMDLYM */ 00014000
/*****************************/ 00015000
/** FUNCTION: CEEDLYM - SUSPENDS ENCLAVE EXECUTION IN MILLISECS **/ 00016000
/** THIS EXAMPLE CALLS CEEDLYM TO SUSPEND PROCESSING OF THE */ 00017000
/** ACTIVE ENCLAVE FOR SPECIFIED NUMBER OF MILLISECS. */ 00018000
/** */ 00019000
/*****************************/ 00020000
PLIDLYM: PROCEDURE OPTIONS (MAIN) REORDER; 00023000
%INCLUDE CEEIBMAW; 00025000
%INCLUDE CEEIBMCT; 00026000
DECLARE MILLISECS REAL FIXED BINARY(31,0); 00028000
DECLARE LILIAN REAL FIXED BINARY(31,0); 00029000
DECLARE SECONDS REAL FLOAT DECIMAL(16); 00030000
DECLARE GREGORN CHARACTER (17); 00031000
DECLARE 01 FC1, /* FEEDBACK TOKEN FOR CEELOCT */ 00033000
03 MSGSEV REAL FIXED BINARY(15,0), 00034000
03 MSGNO REAL FIXED BINARY(15,0), 00035000
03 FLAGS, 00036000
05 CASE BIT(2), 00037000
05 SEVERITY BIT(3), 00038000
05 CONTROL BIT(3), 00039000
03 FACID CHAR(3), 00040000
03 ISI REAL FIXED BINARY(31,0); 00041000
DECLARE 01 FC2, /* FEEDBACK TOKEN FOR CEEDLYM */ 00043000
03 MSGSEV REAL FIXED BINARY(15,0), 00044000
03 MSGNO REAL FIXED BINARY(15,0), 00045000
03 FLAGS, 00046000
05 CASE BIT(2), 00047000
05 SEVERITY BIT(3), 00048000
05 CONTROL BIT(3), 00049000
03 FACID CHAR(3), 00050000
03 ISI REAL FIXED BINARY(31,0); 00051000
CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC1 ); 00053000
IF FBCHECK(FC1, CEE000) THEN DO; 00054000
PUT SKIP LIST ( 'CEEDLYM START DATE AND TIME: ' || GREGORN ); 00055000
END; 00056000
ELSE DO; 00057000
PUT ( 'CEEDLYM FAILED WITH MSG ' || FC1.MSGNO ); 00058000
STOP; 00059000
END; 00060000
MILLISECS = 6000; 00061000
CALL CEEDLYM(MILLISECS, FC2); 00062000
IF FBCHECK(FC2, CEE000) THEN DO; 00063000
CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC1 ); 00065000
IF FBCHECK(FC1, CEE000) THEN DO; 00066000
PUT SKIP LIST ( 'CEEDLYM FINISH DATE AND TIME: ' || GREGORN ); 00067000
END; 00068000
```

Figure 97. PL/I Example of CEEDLYM (Part 1 of 2)
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 CEEPDM 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.

Syntax

```
CEEDSHP(heap_id, fc)
```

heap_id (input)

A fullword binary signed integer. heap_id is a token specifying the discarded heap. A heap_id of 0 is not valid; the initial heap is logically created during enclave initialization and cannot be discarded.

fc (output)

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>0802</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0P2</td>
<td>4</td>
<td>0802</td>
<td>Heap storage control information was damaged.</td>
</tr>
<tr>
<td>CEE0P3</td>
<td>3</td>
<td>0803</td>
<td>The heap identifier in a get storage request or a discard heap request was unrecognized.</td>
</tr>
<tr>
<td>CEE0PC</td>
<td>3</td>
<td>0812</td>
<td>An invalid attempt to discard the Initial Heap was made.</td>
</tr>
</tbody>
</table>

Usage notes

- After the call to CEEDSHP, any existing pointers to storage allocated from this heap are dangling pointers, that is, pointers to storage that is freed. Using these pointers can cause unpredictable results.
- z/OS UNIX considerations—CEEDSHP applies to the enclave. Language Environment frees all storage in the heap regardless of which thread allocated it.

For more information

- For more information about the CEEDSHP callable service, see "CEECRHP—Create new additional heap" on page 227.
For more information about the CEEGTST callable service, and “CEEGTST—Get heap storage” on page 331.

Examples

Figure 98 shows an example of CEEDSHP being called by C/C++.

```c
/*Module/File Name: EDCDSHP */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceedcct.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);
  }
  /* . . . */
}
```

Figure 98. C/C++ Example of CEEDSHP

Figure 99 on page 270 shows an example of CEEDSHP being called by COBOL.
**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-Severity.
          02 1-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 "CEECHRHP" USING HEAPID, HPSIZE,
    INC, OPTS, FC. **
  IF CEE000 of FC THEN
    DISPLAY "Created heap number " HEAPID
    " which is " HPSIZE " bytes long"
  ELSE
    DISPLAY "CEECHRHP failed with msg "
    Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

Figure 99. COBOL Example of CEEDSHP (Part 1 of 2)
**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.

**Figure 100 shows an example of CEEDSHP being called by PL/I.**

*PROCESS MACRO;
  /*Module/File Name: IBMDSHP */
  ***************************************************/
  /**
  /** Function: CEEDSHP - discard heap
  /**
  /** In this example, calls are made to CEECRHP
  /** and CEEDSHP to create a heap of 4096 bytes
  /** and then discard it.
  /**
  /*****************************************************************************/
PLIDSHP: PROC OPTIONS(MAIN);
  %INCLUDE CEEIBMAW;
  %INCLUDE CEEIBMCT;
  DCL HEAPID REAL FIXED BINARY(31,0) ;
  DCL HPSIZE REAL FIXED BINARY(31,0) ;
  DCL INCR REAL FIXED BINARY(31,0) ;
  DCL OPTS REAL FIXED BINARY(31,0) ;
  DCL 01 FC, /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
    03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);
Figure 100. PL/I Example of CEEDSHP (Part 1 of 2)
CEEDYWK—Calculate day of week from Lilian date

CEEDYWK calculates the day of the week on which a Lilian date falls. The day of the week is returned to the calling routine as a number between 1 and 7. The number returned by CEEDYWK is useful for end-of-week calculations.

Syntax

```
CEEDYWK(—input_Lilian_date—,—output_day_no—,—fc—)
```

`input_Lilian_date (input)`
A 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582. For example, 16 May 1988 is day number 148138. The valid range of `input_Lilian_date` is between 1 and 3,074,324 (15 October 1582 and 31 December 9999).

`output_day_no (output)`
A 32-bit binary integer representing `input_Lilian_date's` day-of-week: 1 equals
CEEDYWK

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 Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2EG</td>
<td>3</td>
<td>2512</td>
<td>The Lilian date value passed in a call to CEEDATE or CEEDYWK was not within the supported range.</td>
</tr>
</tbody>
</table>

Usage notes
- z/OS UNIX consideration—In multithread applications, CEEDYWK affects only the calling thread.
- COBOL consideration—The CEEDYWK callable service has different values than the COBOL ACCEPT statement with conceptual data item DAY-OF-WEEK. Use one or the other, not both.

Examples
Figure 101 on page 274 is an example of CEEDYWK called by C/C++. 
Figure 102 on page 275 is an example of CEEDYWK called by COBOL.

```c
/*Module/File Name: EDCDYWK */
#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main (void) {
    _INT4 in_date, day;
    _FEEDBACK fc;

    in_date = 139370; /* Thursday */
    CEEDYWK(&in_date,&day,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEDYWK failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }
    printf("Lilian date %d, occurs on a ",in_date);
    switch(day) {
        case 1: printf("Sunday.\n"); break;
        case 2: printf("Monday.\n"); break;
        case 3: printf("Tuesday.\n"); break;
        case 4: printf("Wednesday.\n"); break;
        case 5: printf("Thursday.\n"); break;
        case 6: printf("Friday.\n"); break;
        case 7: printf("Saturday.\n"); break;
        default: printf(" ERROR! DAY RETURN BY CEEDYWK UNKNOWN\n"); break;
    }
}
```

Figure 101. C/C++ Example of CEEDYWK

Figure 102 on page 275 is an example of CEEDYWK called by COBOL.
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.

Figure 102. COBOL Example of CEEDYWK (Part 1 of 2)
** 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.
CEEENV—Process environmental variables

CEEENV processes environment variables depending on the function code passed in as input. Based on the input to this function, CEEENV can do the following:

- Obtain the value for an existing environment variable
- Create a new environment variable with a value
- Clear all environment variables
- Delete an existing environment variable
- Overwrite the value for an existing environment variable

---

Figure 103. PL/I Example of CEEDYWK

```plaintext
**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 CEEIMAN;
%INCLUDE CEEIBMCT;

DCL LILIAN REAL FIXED BINARY(31,0) ;
DCL DAYNUM REAL FIXED BINARY(31,0) ;
DCL 01 FC, /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
    03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);

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;
```
**CEEENV**

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>⬠CEEENV( function_code, name_length, name, value_length ⬠ ⬠ value, fc )</td>
</tr>
</tbody>
</table>

**function_code (input)**
A fullword binary integer containing the function code of one of the following values:

1. Searches the environment table for environment variables specified by `name` and if found returns a pointer to `value`.
2. Adds an environment variable to the environment table. It does not overwrite an existing environment variable.
3. Clears all environment variables from the environment table.
4. Deletes an environment variable from the environment table.
5. Overwrites an existing environment variable in the environment table and adds the environment variable if it does not exist.

**name_length (input)**
A fullword binary integer containing the length of the name for the environment variable. If the request is for function code 3, this argument is ignored.

**name (input)**
Specifies the name of an environment variable. If the request is for function code 3, this argument is ignored.

**value_length (input/output)**
A fullword binary integer containing the length of the value for the environment variable. This argument is input for setting or modifying an environment variable and is output for getting an environment variable value. If the request is for function code 3 or 4, this argument is ignored.

**value (input/output)**
A field that contains the address of a string that contains the value of the environment variable. This argument is input for setting or modifying an environment variable and is output for getting an environment variable value. If the request is for function code 3 or 4, this argument is ignored. For function code 1 (get), the value address is 0 (NULL) if the name is not found in the environment.

**fc (output)**
The parameter in which the feedback code is placed. The following conditions can result from this service.

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE51O</td>
<td>3</td>
<td>5176</td>
<td>Not enough memory available.</td>
</tr>
<tr>
<td>CEE51P</td>
<td>3</td>
<td>5177</td>
<td>Bad input character detected for name or value.</td>
</tr>
<tr>
<td>CEE51Q</td>
<td>3</td>
<td>5178</td>
<td>Bad address detected for the ENVAR anchor or environment variable array.</td>
</tr>
</tbody>
</table>
Usage notes

- The environment array is searched sequentially and the first occurrence of name is used.
- Because an application can manipulate the environment using the environ pointer, Language Environment cannot guarantee a single instance of any particular environment variable.
- For a function code 1 request, the storage returned for the value character string is supplied by Language Environment. There is one buffer per thread, making it the user’s responsibility to use or save the value before the next call for a function code 1 on that thread.
- Specifying 0 for value_length with function code 2 or 5 results in the environment variable being removed from the environment.
- Applications should not define environment variables that begin with the characters “_BPXK_”, “_EDC_”, or “_CEE_” because there might be conflicts with variable names reserved for z/OS that begin with those characters.
- name and value are copied into storage owned by Language Environment.
- If a function code 1 request is made and the variable name is not found in the environment, value_length is set to 0 upon return.
- A NULL character in name is not valid and causes feedback code CEE51P to be returned.
- A NULL character in value is interpreted as a string terminator. If a NULL character is imbedded, CEEENV will truncate the value string at the last character preceding the NULL.

Examples

Figure 104 on page 280 is an example of CEEENV called by C/C++.
/*Module/File Name: EDCENV */
/*********************************************************************/
/**/
/* THIS EXAMPLE CALLS CEEENV TO CLEAR THE ENVIRONMENT VARIABLE Array, SET AN ENVIRONMENT VARIABLE AND THEN GET THE VARIABLE FROM THE ARRAY. */
/**/
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>

int main(void) {
  _INT4 func_code;
  _INT4 name_len;
  _POINTER name = (_POINTER)malloc(255);
  _INT4 value_len;
  _POINTER value_ptr;
  _FEEDBACK fc;
  char value[255];

  /* Clearing all the environment variables */
  func_code = 3;
  CEEENV(&func_code,&name_len,name,&value_len,&value_ptr,&fc);

  /* Setting a new environment variable */
  func_code = 2;
  strcpy(name,"ENVAR1");
  strcpy(value,"DEFAULT");
  name_len = strlen(name);
  value_len = strlen(value);
  value_ptr = value;
  CEEENV(&func_code,&name_len,name,&value_len,&value_ptr,&fc);

  /* Getting the value of the new variable */
  func_code = 1;
  strcpy(name,"ENVAR1");
  strcpy(value,"" );
  value_ptr = value;
  CEEENV(&func_code,&name_len,name,&value_len,&value_ptr,&fc);

  if (value_len != 0)
    printf("$%s=%s
",name,value_ptr);
  else
    printf("$%s not found\n",name);
}

Figure 104. C/C++ Example of CEEENV

[Figure 105 on page 281] is an example of CEEENV called by COBOL.
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTENV.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 FUNCCODE PIC 9(9) BINARY.
  01 NAMELEN PIC 9(9) BINARY.
  01 VALUELEN PIC 9(9) BINARY.
  01 NAME PIC X(255).
  01 VALPTR POINTER.
  01 FC.
     02 CONDITION-TOKEN-VALUE.
       COPY CEEIGZCT.
       03 CASE-I-CONDITION-ID.
       04 SEVERITY PIC S9(4) BINARY.
       04 MSG-NUM PIC S9(4) BINARY.
       03 CASE-SEV-CTL PIC X.
       03 FACILITY-ID PIC XXX.
       02 I-S-INFO PIC S9(9) BINARY.
  01 VAL PIC X(255).

LINKAGE SECTION.
  01 VAR PIC X(5000).

PROCEDURE DIVISION.
MAIN-PROG.

******************************************************************************
** Function: CEEENV - Process environment variables
**
** This example calls CEEENV to clear the environment variable array, set an
** environment variable and then get the variable from the array.
**
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTENV.

DATA DIVISION.
WORKING-STORAGE SECTION.

******************************************************************************
** Clear environment array
******************************************************************************
   MOVE 3 TO FUNCCODE.
   CALL "CEEENV" USING FUNCCODE,
                NAMELEN,
                NAME,
                VALUELEN,
                VALPTR,
                FC.

******************************************************************************

Figure 105. COBOL Example of CEEENV (Part 1 of 2)
** Set an environment variable
****************************
MOVE 2 TO FUNCCODE.
MOVE "ENVAR1" TO NAME.
MOVE "DEFAULT" TO VAL.
MOVE 6 TO NAMELEN.
MOVE 7 TO VALUELEN.
SET VALPTR TO ADDRESS OF VAL.
CALL "CEEENV" USING FUNCCODE,
    NAMELEN,
    NAME,
    VALUELEN,
    VALPTR,
    FC.

****************************
** Get the environment variable
****************************
MOVE 1 TO FUNCCODE.
MOVE " " TO VAL.
MOVE 0 TO VALUELEN.
CALL "CEEENV" USING FUNCCODE,
    NAMELEN,
    NAME,
    VALUELEN,
    VALPTR,
    FC.

IF VALUELEN NOT = 0 THEN
    SET ADDRESS OF VAR TO VALPTR
    DISPLAY NAME(1:NAMELEN) "=" VAR(1:VALUELEN)
ELSE
    DISPLAY NAME ' NOT FOUND'
END-IF.
GOBACK.

Figure 105. COBOL Example of CEEENV (Part 2 of 2)

Figure 106 on page 283 is an example of CEEENV called by PL/I.
**Module/File Name: IBMENV**

**Function: CEEENV - process environment variables**

This example calls CEEENV to clear the environment variable array, set an environment variable and then get the variable from the array.

---

```pli
#include CEEIBMAW;
#include CEEIBMCT;

DECLARE FUNC_CODE REAL FIXED BINARY(31,0);
DECLARE NAME_LEN REAL FIXED BINARY(31,0);
DECLARE VALUE_LEN REAL FIXED BINARY(31,0);
DECLARE NAME CHAR(255) INIT('');
DECLARE VALUE CHAR(255) INIT('');
DECLARE VALUE2 CHAR(255) BASED(VALUE_PTR);
DECLARE VALUE_PTR POINTER;

DECLARE 01 LE_FEEDBACK_CODE,
  03 MSGSEV REAL FIXED BINARY(15,0),
  03 MSGNO REAL FIXED BINARY(15,0),
  03 FLAGS,
  05 CASE BIT(2),
  05 SEVERITY BIT(3),
  05 CONTROL BIT(3),
  03 FACID CHAR(3),
  03 ISI REAL FIXED BINARY(31,0);

DECLARE MSG_STRING CHAR(255) VARYING;
DECLARE MSG_DEST REAL FIXED BINARY(31,0);
DECLARE SUBSTR BUILTIN;
DECLARE SYSPRINT FILE PRINT;

MSG_DEST = 2;

/******************************************
/* Clear all the environment variables */
/******************************************
FUNC_CODE = 3;
call CEEENV(FUNC_CODE,
  NAME_LEN,
  NAME,
  VALUE_LEN,
  VALUE_PTR,
  LE_FEEDBACK_CODE);
```

---

Figure 106. PL/I Example of CEEENV (Part 1 of 2)
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 CEE3CTY callable service, not the CEESETL callable service or the setlocale() function.

Syntax

```
>>>CEEFMDA(---country_code---,---date_pic_str---,---fc---)
```

country_code (input)

A 2-character fixed-length string representing one of the country codes found in Table 32 on page 515. The 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 CEE3CTY callable service) is used. If you specify a country_code that is not valid, the default date format is 'YYYY-MM-DD'.

date_pic_str (output)

A fixed-length 80-character string (VSTRING), returned to the calling routine. It contains the default date picture string for the country specified. The picture string is left-justified and padded on the right with blanks if necessary.
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3CB</td>
<td>2</td>
<td>3467</td>
<td>The country code country_code was invalid for CEEFMDA. The default date picture string date_pic_string was returned.</td>
</tr>
</tbody>
</table>

Usage note

- z/OS UNIX considerations—CEEFMDA applies to the enclave. Every enclave has a single current country setting that has a single date format. Every thread in every enclave has the same default.

For more information

- For a list of the default settings for a specified country, see Table 32 on page 515
- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 132 for an explanation of the CEE3CTY callable service.

Examples

Figure 107 is an example of CEEFMDA called by C/C++.

```c
/*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);
}
```

Figure 107. C/C++ Example of CEEFMDA

Figure 108 on page 286 is an example of CEEFMDA called by COBOL.
Figure 108. COBOL Example of CEEFMDA

Figure 109 on page 287 is an example of CEEFMDA called by PL/I.
CEEFMDT—Get default date and time format

CEEFMDT returns the default date and time picture strings for the country specified by country_code. CEEFMMDT is affected only by the country code setting of the COUNTRY run-time option or CEE3CTY callable service, not the CEESETL callable service or the setlocale() function.

Syntax

```
  CEEFMMDT(country_code, datetime_str, fc)
```

**country_code (input)**

A 2-character fixed-length string representing one of the country codes found in Table 32 on page 515. The 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 CEE3CTY) is used. If you specify a country_code that is not valid, the default date/time picture string is 'YYYY-MM-DD HH:MI:SS'.

**datetime_str (output)**

A fixed-length 80-character string (VSTRING), returned to the calling routine. It
contains the default date and time picture string for the country specified. The picture string is
left-justified and padded on the right with blanks, if necessary.

\[ fc \] (output)
A 12-byte feedback code, optional in some languages, that indicates the result
of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3CF</td>
<td>2</td>
<td>3471</td>
<td>The country code ( \text{country_code} ) was invalid for CEEFMDT. The default date and time picture string ( \text{datetime_str} ) was returned.</td>
</tr>
</tbody>
</table>

Usage note
- \( z/OS \) UNIX considerations—CEEFMDT applies to the enclave. Every enclave has a single current
  country setting that has a single date and time format. Every thread in every enclave has the same default.

For more information
- For a list of the default settings for a specified country, see Table 32 on page 515.
- See \"COUNTRY\" on page 22 for an explanation of the COUNTRY run-time option.
- See \"CEE3CTY—Set default country\" on page 132 for an explanation of CEE3CTY.

Examples
Figure 110 is an example of CEEFMDT called by C/C++.

```c
/*Module/File Name: EDCFMDT */
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
  _FEEDBACK fc;
  _CHAR8 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);
}
```

Figure 110. C/C++ Example of CEEFMDT

Figure 111 on page 289 is an example of CEEFMDT called by COBOL.
CEEFMDT

CBL LIB,QUOTE
*Module/File Name: IGZTFMDT
**********************************************************************
** **
** CBLFMDT - Call CEEFMDT to obtain the **
** default date & time format **
**
************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLFMDT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 COUNTRY PIC X(2).
01 PICSTR PIC X(80).
01 FC.  
  02 Condition-Token-Value.
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLFMDT.
** Specify country code for the US
  MOVE "US" TO COUNTRY.
** Call CEEFMDT to return the default date and
** time format for the US
  CALL "CEEFMDT" USING COUNTRY , PICSTR , FC.
** If CEEFMDT runs successfully, display result.
  IF CEE000 of FC THEN
    DISPLAY "The default date and time "
    "format for the US is: " PICSTR
  ELSE
    DISPLAY "CEEFMDT failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.
GOBACK.

Figure 111. COBOL Example of CEEFMDT

Figure 112 on page 290 is an example of CEEFMDT called by PL/I.
CEEFMON—Format monetary string

CEEFMON, analogous to the C function `strfmon()`, converts numeric values to monetary strings according to the specifications passed to it. These specifications work in conjunction with the format conversions set in the current locale. The current locale's LC_MONETARY category affects the behavior of this service, including the monetary radix character, the thousands separator, the monetary grouping, the currency symbols (national and international), and formats.

CEEFMON is sensitive to the locales set by `setlocale()` or `CEESETL`, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```
-- CEEFMON--(omitted_parm---stringin---maxsize---format---)
```

---stringout---result---fc---

---

Figure 112. PL/I Example of CEEFMDT
omitted_parm
This parameter is reserved for future expansion and must be omitted. For information on how to code an omitted parm, see “Invoking callable services” on page 112.

stringin (input)
An 8-byte field that contains the value of a double-precision floating point number.

maxsize (input)
A 4-byte integer that specifies the maximum number of bytes (including the string terminator) that can be placed in stringout.

format (input)
A halfword length-prefixed character string (VSTRING) of 256 bytes that contains plain characters and a conversion specification. Plain characters are copied to the output stream. Conversion specification results in the fetching of the input stringin argument that is converted and formatted.

A conversion specification consists of a percent character (%), optional flags, optional field width, optional left precision, optional right precision, and a required conversion character that determines the conversion to be performed.

Flags (optional)
You can specify one or more of the following flags to control the conversion.

= f
An = followed by a single character f specifies that the character f should be used as the numeric fill character. The default numeric fill character is the space character. This option does not affect the other fill operations (such as field-width filling), which always use the space as the fill character.

^
Do not format the currency amount with the grouping characters. The default is to insert the grouping characters if defined for the current locale.

+ or ( Specifies the style of representing positive and negative currency amounts. You must specify either + or . If + is specified, the locale’s equivalent of + and - are used (for example, in USA: the empty (null) string if positive and - (minus sign) if negative). If ( is specified, the locale’s equivalent of enclosing negative amounts within a parenthesis is used. If this option is not included, a default specified by the current locale is used.

!
Suppresses the currency symbol from the output conversion.

Field Width (optional)
A decimal digit string w that specifies a minimum field width in which the result of the conversion is right-justified (or left-justified if -w is specified).

Left Precision (optional)
A # (pound sign) followed by the decimal digit string n, ( specified as #n), indicates a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to the CEEFMON service aligned in the same columns. It can also be used to fill unused positions with a special character as in $***123.45. This option causes an amount to be formatted as if it has the number of digits specified by n. If more digit
positions are required than are specified, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character. See the $f$ specification above.

If grouping is enabled, it is applied to the combined fill characters plus regular digits. However, if the fill character is not 0 (digit zero), grouping separators inserted after a fill character are replaced by the same fill character ($0,001,234.56$ versus $00001,234.56$).

**Right Precision (optional)**
A period (.) followed by the decimal digit string $p$, (specified as $.p$), indicates the number of digits after the radix character. If the value of the precision $p$ is zero, no radix character appears. If this option is not included, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting.

**Conversion Characters (required)**
One of the following conversion characters must be specified.

- **i** The double argument is formatted according to the locale's international currency format (for example, in USA: USD 1,234.56).
- **n** The double argument is formatted according to the locale's national currency format (for example, in USA: $1,234.56$).
- **%** No argument is converted; the conversion specification $%$ is replaced by a single $%$.

**stringout (output)**
Contains the output of the CEEFMON service.

**result (output)**
Contains the number of characters placed in stringout, if successful. If unsuccessful, an error is reported.

**fc (output/optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
<tr>
<td>CEE3VM</td>
<td>3</td>
<td>4086</td>
<td>Input Error: The number of characters to be formatted must be greater than zero.</td>
</tr>
</tbody>
</table>

**Usage notes**
- PL/I MTF consideration—CEEFMON is not supported in PL/I MTF applications.
- This callable service uses the C/C++ run-time library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

**For more information**
- For more information about the setlocale() function, see "COUNTRY" on page 22, "CEE3CTY—Set default country" on page 132, and "CEE3LNG—Set national language" on page 174.
Examples

Figure 113 is an example of CEEFMON called by COBOL.

CBL LIB,QUOTE
*Module/File Name: IGZTFMON
******************************************************************************
* Example for callable service CEEFMON *
* Function: Convert a numeric value to a *</n> monetary string using specified *
> format passed as parameter. *
> Valid only for COBOL for MVS & VM Release 2 *
> or later. *
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. COBFMON.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 Monetary COMP-2.
 01 Max-Size PIC S9(9) BINARY.
 01 Format-Mon.
    02 FM-Length PIC S9(4) BINARY.
    02 FM-String PIC X(256).
 01 Output-Mon.
    02 OM-Length PIC S9(4) BINARY.
    02 OM-String PIC X(60).
 01 Length-Mon PIC S9(9) BINARY.
 01 Locale-Name.
    02 LN-Length PIC S9(4) BINARY.
    02 LN-String PIC X(256).
** Use Locale category constants
COPY CEEIGZLC.
 01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.
**
PROCEDURE DIVISION.
** Set up locale name for United States
  MOVE 14 TO LN-Length.
  MOVE "En_US.IBM-1047"
> TO LN-String (1:LN-Length).
** Set all locale categories to United States.
** Use LC-ALL category constant from CEEIGZLC.
  CALL "CEESETL" USING Locale-Name, LC-ALL, FC.
** Check feedback code
  IF Severity > 0
    DISPLAY "Call to CEESETL failed." Msg-No
  STOP RUN
END-IF.

Figure 113. COBOL Example of CEEFMON (Part 1 of 2)
**Figure 114** is an example of CEEFMON called by PL/I.

**Figure 113. COBOL Example of CEEFMON (Part 2 of 2)**

```cobol
*PROCESS MACRO;
/*Module/File Name: IBMFMON */
/*****************************/
/* Example for callable service CEEFMON */
/* Function: Convert a numeric value to a monetary */
/* string using specified format passed as parm */
/*****************************/
PLIFMON: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs */
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants */
%INCLUDE CEEIBMLC; /* Locale category constants */
/* CEESETL service call arguments */
DCL LOCALE_NAME CHAR(14) VARYING;
/* CEEFMON service call arguments */
DCL MONETARY REAL FLOAT DEC(16); /* input value */
DCL MAXSIZE_FMON BIN FIXED(31); /* output size */
DCL FORMAT_FMON CHAR(256) VARYING; /* format spec */
DCL RESULT_FMON BIN FIXED(31); /* result status */
DCL OUTPUT_FMON CHAR(60) VARYING; /* output string */
DCL 01 FC, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,,
  05 Case BIT(2),
  05 Severity BIT(3),
  05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
       REAL FIXED BINARY(31,0);
/* init locale name to United States */
LOCALE_NAME = 'En_US.IBM-1047';
/* use LC_ALL category constant from CEEIBMLC */
CALL CEESETL (LOCALE_NAME, LC_ALL, FC);
/* FBCHECK macro used (defined in CEEIBMCT) */
IF FBCHECK (FC, CEE2KE) THEN
  DO; /* invalid locale name */
    DISPLAY ('Locale LC_ALL Call '..FC.MsgNo);
    STOP;
  END;
END.

**Figure 114. PL/I Example of CEEFMON (Part 1 of 2)**

```
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 515. CEEFMTM is affected only by the country code setting of the COUNTRY run-time option or CEE3CTY callable service, not the CEESETL callable service or the setlocale() function.

Syntax

```c
CEEFMTM (country_code, time_pic_str, fc)
```

**country_code** (input)

A 2-character fixed-length string representing one of the country codes found in Table 32 on page 515. The 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 CEE3CTY callable service) is used. If you specify a country_code that is not valid, the default time picture string is 'HH:MI:SS'.

**time_pic_str** (output)

A fixed-length 80-character string (VSTRING), representing the default time picture string for the country specified. The picture string is left-justified and padded on the right with blanks if necessary.

**fc** (output)

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>
### Usage notes

- **z/OS UNIX considerations**—CEEFMTM applies to the enclave. Every enclave has a single current country setting that has a single time format. Every thread in every enclave has the same default.

### For more information

- See “COUNTRY” on page 22 for an explanation of the COUNTRY run-time option.
- See “CEE3CTY—Set default country” on page 132 for an explanation of the CEE3CTY callable service.

### Examples

Figure 115 is an example of CEEFMTM called by C/C++.

```c
/*Module/File Name: EDCFMTM */
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
  _FEEDBACK fc;
  _CHAR2 country;
  _CHAR60 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("%.60s\n",time_pic);
}
```

**Figure 115. C/C++ Example of CEEFMTM**

Figure 116 on page 297 is an example of CEEFMTM called by COBOL.
**Figure 116. COBOL Example of CEEFMTM**

**Figure 117 on page 298** is an example of CEEFMTM called by PL/I.
CEEFRST—Free heap storage

CEEFRST frees storage previously allocated by CEEGTST or by a language intrinsic function. Normally, you do not need to call CEEFRST because Language Environment automatically returns all heap storage to the operating system when the enclave terminates. However, if you are allocating a large amount of heap storage, you should free the storage when it is no longer needed. This freed storage then becomes available for later requests for heap storage, thus reducing the total amount of storage needed to run the application.

All requests for storage are conditional. If storage is not available, the feedback code (fc) is set and returned to you, but the thread does not abend. An attempt to free storage that was already marked as free produces no action and returns a non-CEE000 symbolic feedback code. An attempt to free storage at anything other than a valid starting address produces no action and returns a non-CEE000 symbolic feedback code. The application does not abend.

Figure 117. PL/I Example of CEEFMTM

```pli
/* Process Macro; */
/* Module/FileName: IBMFMTM */
/*****************************************/
/* Function: CEEFMTM - obtain default time format */
/*****************************************/
/* This example calls CEEFMTM to request the default time format for the US and print it out. */
/*****************************************/
PLIFMTM: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;
DCL COUNTRY CHARACTER(2) ;
DCL PICSTR CHAR(80);
DCL 01 FC, /* Feedback token */
     03 MsgSev REAL FIXED BINARY(15,0),
     03 MsgNo REAL FIXED BINARY(15,0),
     03 Flags,
     05 Case BIT(2),
     05 Severity BIT(3),
     05 Control BIT(3),
     03 FacID CHAR(3), /* Facility ID */
     03 ISI /* Instance-Specific Information */
        REAL FIXED BINARY(31,0);
COUNTRY = 'US'; /* Specify country code for the United States */
/* Call CEEFMTM to get default time format for the US */
CALL CEEFMTM ( COUNTRY , PICSTR , FC );
/* Print the default time format for the US */
IF FBCHECK( FC, CEE000) THEN DO;
   PUT SKIP LIST('The default time format for the US is '||PICSTR);
   END;
ELSE DO;
   DISPLAY( 'CEEFMTM failed with msg '||FC.MsgNo );
   STOP;
   END;
END PLIFMTM;
```

298  z/OS V1R13.0 Language Environment Programming Reference
However, if you call CEEFRST for an invalid address, and you had specified TRAP(OFF), your application can abend. Language Environment’s reaction to this is undefined. Also, partial freeing of an allocated area is not supported.

When storage is allocated by CEELST, its allocated size is used during free operations. Storage allocated by CEELST, but not explicitly freed, is automatically freed at enclave termination.

CEEFST 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 CEERHP 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

```
CEEFST(-address-, -fc-)
```

**address (input)**

A fullword address pointer. address is the address returned by a previous CEELST 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 Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0P2</td>
<td>4</td>
<td>0802</td>
<td>Heap storage control information was damaged.</td>
</tr>
<tr>
<td>CEE0PA</td>
<td>3</td>
<td>0810</td>
<td>The storage address in a free storage (CEEFST) request was not recognized, or heap storage (CEECZST) control information was damaged.</td>
</tr>
</tbody>
</table>

**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 CEELST call. If the pointer is not reassigned, any further use of it causes unpredictable results.
z/OS UNIX considerations—CEEFRST applies to the enclave. One thread can allocate storage, and another can free it.

For more information

- See [“CEEGTST—Get heap storage” on page 331](#) for more information about the CEEGTST callable service.
- See [“HEAP” on page 36](#) for further information about the HEAP run-time option.
- See [“CEECRHP—Create new additional heap” on page 227](#) for more information about the CEECRHP callable service.
- See [“STORAGE” on page 75](#) for further information about the STORAGE run-time option.

Examples

Figure 118 is an example of CEEFRST called by C/C++.

```c
/*Module/File Name: EDCFRST */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
  _INT4 heapid, size;
  _POINTER address;
  _FEEDBACK fc;
  /* */
  . /* */
  _INT4 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);
  }
  /* */
  . /* */
}
```

Figure 118. C/C++ Example of CEEFRST

Figure 119 on page 301 is an example of CEEFRST called by COBOL.
**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.

```cobol
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.
    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.
```

Figure 119. COBOL Example of CEEFRST

Figure 120 on page 302 is an example of CEEFRST called by PL/I.
CEEFRST

+PROCESS MACRO;
/*Module/File Name: IBMFRT */
*******************************************************************************/
/**
** 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 CEEIIBMW;
%INCLUDE CEEIIBMC;
DCL ADDRSS POINTER;
DCL HEAPID REAL FIXED BINARY(31,0);
DCL STGSIZE REAL FIXED BINARY(31,0);
DCL 01 FC, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
  05 Case BIT(2),
  05 Severity BIT(3),
  05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);
DCL 01 FC2, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
  05 Case BIT(2),
  05 Severity BIT(3),
  05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);
HEAPID = 0; /* get storage from the initial heap */
STGSIZE = 4000; /* get 4000 bytes of storage */
/* Call CEEGTST to obtain the storage */
CALL CEEGTST ( HEAPID, STGSIZE, ADDRSS, FC );
IF FBCHECK( FC, CEE000) THEN DO;
  PUT SKIP LIST( 'Obtained ' || STGSIZE
    || ' bytes of storage at location ' ||
    DECIMAL( UNSPEC( ADDRSS ) )
    || ' from heap ' || HEAPID );
END;
ELSE DO;
  DISPLAY( 'CEEGTST failed with msg ' || FC.MsgNo);
  STOP;
END;

Figure 120. PL/I Example of CEEFRST (Part 1 of 2)
CEEFTDS

/* Call CEEFRST with the address returned from */
/* CEEGST to free the storage allocated by */
/* the call to CEEGST */
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;

Figure 120. PL/I Example of CEEFRST (Part 2 of 2)

CEEFTDS—Format time and date into character string

CEEFTDS, analogous to the C function strftime(), converts the internal time and date to character strings according to the specifications passed to it by the TD_STRUCT. These specifications work in conjunction with the format conversions set in the current locale. The current locale’s LC_TIME category affects the behavior of this service, including the actual values for the format specifiers. CEEFTDS is sensitive to the locales set by setlocale() or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Syntax

```plaintext
CEEFTDS(omitted_parm, time_and_date, maxsize, format, stringout, fc)
```

omitted_parm

This parameter is reserved for future expansion and must be omitted. For information on how to code an omitted parm, see "Invoking callable services" on page 112.

time_and_date (input)

Points to the time structure that is to be converted. The structure has the following format:

- **td_sec**
  A 4-byte integer that describes the number of seconds in the range of 0 through 59.

- **td_min**
  A 4-byte integer that describes the number of minutes in the range of 0 through 59.

- **td_hour**
  A 4-byte integer that describes the number of hours in the range of 0 through 23.

- **td_mday**
  A 4-byte integer that describes the day of the month in the range of 1 through 31.
**td_mon**
A 4-byte integer that describes the month of the year in the range of 0 through 12.

**td_year**
A 4-byte integer that describes the year of an era.

**td_wday**
A 4-byte integer that describes the day of the week in the range of 0 through 6.

**td_yday**
A 4-byte integer that describes the day of the year in the range of 0 through 365.

**td_isdst**
A 4-byte integer that is a flag for daylight savings time.

**maxsize** *(input)*
A 4-byte integer that specifies the maximum length (including the string terminator) of the string that can be placed in *stringout*.

**format** *(input)*
A halfword length-prefixed character string (VSTRING) of 256 bytes that contains the conversion specifications. The format parameter is a character string containing two types of objects: plain characters that are placed in the output string and conversion specifications that convert information from *time_and_date* into readable form in the output string. Each conversion specification is a sequence of this form:

```
%[-][width][.precision]type
```

%  A percent sign (%) introduces a conversion specification. This character is required.

**width** *(optional)*
An optional decimal digit string that specifies a minimum field width. A converted value that has fewer characters than the field width is padded with spaces to the left. If the decimal digit string is preceded by a minus sign, padding with spaces occurs to the right of the converted value.

If no width is given, for numeric fields the appropriate default width is used with the field padded on the left with zeros, as required. For strings, the output field is made exactly wide enough to contain the string.

**precision** *(optional)*
An optional value that specifies the maximum number of characters to be printed for the conversion specification. The precision value is a decimal digit string preceded by a period. If the value to be output is longer than the precision, it is truncated on the right.

**type**
Specifies the type of conversion to be performed. The characters and their meanings are:

- `%a`  The abbreviated weekday name (for example, Sun) defined by the *abday* statement in the current locale.

- `%A`  The full weekday name (for example, Sunday) defined by the *day* statement in the current locale.

- `%b`  The abbreviated month name (for example, Jan) defined by the *abmon* statement in the current locale.
%B  The full month name (for example, January) defined by the month statement in the current locale.
%c  The date and time format defined by the d_t_fmt statement in the current locale.
%d  The day of the month as a decimal number (01 to 31).
%D  The date in %m/%d/%y format (for example, 01/31/91).
%e  The day of the month as a decimal number (1 to 31). The %e field descriptor uses a two-digit field. If the day of the month is not a two-digit number, the leading digit is filled with a space character.
%E  The combined alternative era year and name, respectively, in %o %N format in the current locale.
%F  The ISO 8601:2000 standard date format, equivalent to %Y-%m-%d.
%g  The last 2 digits of the week-based year as a decimal number [00,99].
%G  The week-based year as a 4-digit decimal number.
%h  The abbreviated month name (for example, Jan) defined by the abmon statement in the current locale. This field descriptor is a synonym for the %b field descriptor.
%H  The 24-hour clock hour as a decimal number (00 to 23).
%I  The 12-hour clock hour as a decimal number (01 to 12).
%j  The day of the year as a decimal number (001 to 366).
%m  The month of the year as a decimal number (01 to 12).
%M  The minutes of the hour as a decimal number (00 to 59).
%n  Specifies a new-line character.
%N  The alternative era name in the current locale.
%o  The alternative era year in the current locale.
%p  The AM or PM string defined by the am_pm statement in the current locale.
%r  The 12-hour clock time with AM/PM notation as defined by the t_fmt_ampm statement (%I:%M:%S [AM|PM]) in the current locale.
%S  The seconds of the minute as a decimal number (00 to 60).
%t  Specifies a tab character.
%T  Represents 24-hour clock time in the format %H:%M:%S (for example, 16:55:15).
%U  The week of the year as a decimal number (00 to 53). Sunday, or its equivalent as defined by the day statement, is considered the first day of the week for calculating the value of this field descriptor.
%w  The day of the week as a decimal number (0 to 6). Sunday, or its equivalent as defined by the .day statement, is considered as 0 for calculating the value of this field descriptor.
CEEFTDS

%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 dFmt statement in the current locale.

%X The time format defined by the tFmt statement in the current locale.

%y The year of the century (00 to 99).

%Y The year as a decimal number (for example, 1989).

%z The offset from UTC in ISO8601:2000 standard format (+hhmm or -hhmm). For example, “-0430” means 4 hours 30 minutes behind UTC (west of Greenwich).

%Z The time-zone name, if one can be determined (for example, EST); no characters are displayed if a time zone cannot be determined.

%% Specifies a percent sign (%) character.

stringout (output)

A halfword length-prefixed character string (VSTRING) of 256 bytes that contains the formatted time and date output of the CEEFTDS service.

fc (output/optional)

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following feedback codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
<tr>
<td>CEE3VM</td>
<td>3</td>
<td>4086</td>
<td>Input Error: The number of characters to be formatted must be greater than zero.</td>
</tr>
</tbody>
</table>

Usage notes

- PL/I MTF consideration—CEEFTDS is not supported in PL/I MTF applications.
- This callable service uses the C/C++ run-time library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- For more information about the setlocale() function, see “COUNTRY” on page 22, “CEE3CTY—Set default country” on page 132, and “CEE3LNG—Set national language” on page 174.
- For more information about the CEESETL callable service, see “CEESETL—Set locale operating environment” on page 448.

Examples

Figure 121 on page 307 is an example of CEEFTDS called by COBOL.
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINFTDS.
DATA DIVISION.
WORKING-STORAGE SECTION.
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.
COPY CEEIGZLC.

01 Ptr-FTDS  POINTER.
01 Output-FTDS.
  02 O-Length PIC S9(4) BINARY.
  02 O-String PIC X(72).
01 Format-FTDS.
  02 F-Length PIC S9(4) BINARY.
  02 F-String PIC X(64).
01 Max-Size PIC S9(9) BINARY.
01 FC.
  02 Condition-Token-Value.
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No  PIC S9(4) BINARY.
  03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
02 I-S-Info PIC S9(9) BINARY.

LINKAGE SECTION.
* Use TD-Struct for calls to CEEFTDS
COPY CEEIGZTD.

Figure 121. COBOL Example of CEEFTDS (Part 1 of 2)
Figure 121. COBOL Example of CEEFTDS (Part 2 of 2)

Figure 122 is an example of CEEFTDS called by PL/I.

### PL/I Example of CEEFTDS (Part 1 of 2)

```plaintext
* 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 */
```
CEEGMT—Get current Greenwich Mean Time

CEEGMT returns the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. The returned values are compatible with those generated and used by the other Language Environment date and time services. In order for the results of this service to be meaningful, your system’s TOD (time-of-day) clock must be set to Greenwich Mean Time and be based on the standard epoch. Use CEEGMT0 to get the offset from GMT to local time.

The values returned by CEEGMT are handy for elapsed time calculations. For example, you can calculate the time elapsed between two calls to CEEGMT by calculating the differences between the returned values.

Language Environment treats Coordinated Universal Time (UTC) and Greenwich Mean Time (GMT) as the same. You can use the CEEUTC service, which is an alias of the CEEGMT service, to get the same value.

**Syntax**

```
>>> CEEGMT(output_GMT_Lilian, output_GMT_seconds, fc)
```
**CEEDGMT**

**output GMT Lilian (output)**

A 32-bit binary integer representing the current time in Greenwich, England, in the Lilian format (the number of days since 14 October 1582). For example, 16 May 1988 is day number 148138. If GMT is not available from the system, `output GMT Lilian` is set to 0 and CEEGMT terminates with a non-CEE000 symbolic feedback code.

**output GMT_seconds (output)**

A 64-bit double floating-point number representing the current date and time in Greenwich, England, as the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). 19:00:01.078 on 16 May 1988 is second number 12,799,191,601.078. If GMT is not available from the system, `output GMT_seconds` is set to 0 and CEEGMT terminates with a non-CEE000 symbolic feedback code.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2E6</td>
<td>3</td>
<td>2502</td>
<td>The UTC/GMT was not available from the system.</td>
</tr>
</tbody>
</table>

**Usage notes**

- CEEDATE converts `output GMT Lilian` to a character date, and CEEDATM converts `output GMT_seconds` to a character timestamp.
- CICS consideration—CEEDGMT does not use the OS TIME macro.
- z/OS UNIX consideration—In multithread applications, CEEGMT affects only the calling thread.
- Setting the TOD (time-of-day) clock to anything before January 1, 1972 may produce unpredictable results in your applications.

**For more information**

- See “CEEDGTO—Get offset from Greenwich Mean Time to local time” on page 313 for more information about the CEEGTO callable service.

**Examples**

Figure 123 on page 311 is an example of CEEGMT called by C/C++.
/*Module/File Name: EDCGMT */
#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {

_FEEDBACK fc;
_INT4   lilGMT_date;
_FLOAT8 secGMT_date;

CEEGMT(&lilGMT_date,&secGMT_date,&fc);
if (_FBCHECK (fc , CEE000 ) !=0) {
   printf("CEEGMT failed with message number %d
",
      fc.tok_msgno);
   exit(2999);
}

printf("The current Lilian date in Greenwich,");
printf(" England is %d
", lilGMT_date);
}

Figure 123. C/C++ Example of CEEGMT

Figure 124 on page 312 is an example of CEEGMT called by COBOL.
CBL Lib,Quote

*Module/File Name: IGZTGMT
*************************************************
** **
** IGZTGMT - Call CEEGMT to get current **
** Greenwich Mean Time **
** **
** In this example, a call is made to CEEGMT **
** to return the current GMT as a Lilian date **
** and as Lilian seconds. The results are **
** displayed. **
** **
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGMT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 LILIAN PIC S9(9) BINARY.
 01 SECS COMP-2.
 01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-5-Info
      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.

Figure 124. COBOL Example of CEEGMT

Figure 125 on page 313 is an example of CEEGMT called by PL/I.
CEEGMTO

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

```bash
>>> CEEGMTO(–offset_hours–,–offset_minutes–,–offset_seconds–,–fc–)
```

`offset_hours` (output)
A 32-bit binary integer representing the offset from GMT to local time, in hours. For example, for Pacific Standard Time, `offset_hours` equals −8. If local time offset is not available, `offset_hours` equals 0 and CEEGMT 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 CEEGMT 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, CEEGMT would return −28,800 (−8 * 60 * 60). offset_seconds can be used with CEEGMT to calculate local date and time. See “CEEGMT—Get current Greenwich Mean Time” on page 309 for more information.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2E7</td>
<td>3</td>
<td>2503</td>
<td>The offset from UTC/GMT to local time was not available from the system.</td>
</tr>
</tbody>
</table>

Usage notes
• CEEDATM is used to convert number of seconds to a character timestamp.
• CICS consideration—CEEGMT does not use the OS TIME macro.
• z/OS UNIX consideration—In multithread applications, CEEGMT affects only the calling thread.

For more information
• See “CEEDATM—Convert seconds to character timestamp” on page 243 for more information about the CEEDATM callable service.

Examples
Figure 126 is an example of CEEGMT called by C/C++.

```c
/*Module/File Name: EDCGMT */
#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc;
    _INT4 GMT_hours,GMT_mins;
    _FLOAT8 GMT_secs;
    CEEGMT(&GMT_hours,&GMT_mins,&GMT_secs,&fc);
    if ( _FBCHECK ( fc , CEE000 ) !=0 ) {
        printf("CEEGMT 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);
}
```

Figure 126. C/C++ Example of CEEGMT
Figure 127 is an example of CEEGMT called by COBOL.

---

CEEGMTO

CBL LIB,QUOTE

*Module/File Name: IGZTGMTO

*************************************************

** IGZTGMTO - Call CEEGMT to get offset from Greenwich Mean Time to local time **

** In this example, a call is made to CEEGMT 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 CEEIZCT.
      03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
PARA-CBLGMTO.
   CALL "CEEGMTO" USING HOURS, MINUTES, SECONDS, FC.
   IF CEE000 of FC THEN
      DISPLAY "Local time differs from GMT "
      "by: " HOURS " hours, "
      MINUTES " minutes, and "
      SECONDS " seconds. "
   ELSE
      DISPLAY "CEEGMTO failed with msg "
      Msg-No of FC UPON CONSOLE
      STOP RUN
   END-IF.
GOBACK.

---

Figure 127. COBOL Example of CEEGMT (Part 1 of 2)

---

Figure 127. COBOL Example of CEEGMT (Part 2 of 2)

Figure 128 on page 316 is an example of CEEGMT called by PL/I.
CEEGPID—Retrieve the Language Environment version and platform ID

CEEGPID retrieves the Language Environment version ID and the platform ID of the version and platform of Language Environment that is processing the currently active condition.

Syntax

```
CEEGPID(CEE_Version_ID, Plat_ID, fc)
```

**CEE_Version_ID (output)**

A four-byte hexadecimal number representing the version of Language Environment that created this data block.

```
[pp]vv|rr|mm
```

- `pp` = Product Number
- `vv` = Version
- `rr` = Release
- `mm` = Modification

---

Figure 128. PL/I Example of CEEGMTO
CEEGPID

The current value of this parameter is:

- X'04010800'  Version 1 Release 8 Modification 0
- X'04010900'  Version 1 Release 9 Modification 0
- X'04010A00'  Version 1 Release 10 Modification 0
- X'04010B00'  Version 1 Release 11 Modification 0
- X'04010C00'  Version 1 Release 12 Modification 0
- X'04010D00'  Version 1 Release 13 Modification 0

Plat_ID (output)
A fullword integer representing the platform used for processing the current condition. The current values of this parameter are:

- 3  z/OS or VM
- 4  AS/400®

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

Usage notes
- z/OS UNIX consideration—In multithread applications, CEEGPID affects only the calling thread.
- To make a decision concerning Language Environment and z/OS levels at assembly time instead of run-time, use CEEGLOB instead. See "z/OS Language Environment Programming Guide" for more information.

Examples
Figure 129 on page 318 is an example of CEEGPID called by C/C++.
Figure 129. C/C++ Example of CEEGPID

Figure 130 on page 319 is an example of CEEGPID called by COBOL.
CEEGPID

** Module/File Name: IGZTGPID  
**************************  
**  ** IGZTGPID - Call CEEGPID to retrieve the  **  
**  LE version and platform ID  **  
**************************

IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGPID.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 VERSION PIC S9(9) BINARY.
01 VER-MAP REDEFINES VERSION.
   03 FILLER PIC X(1).
   03 VER-VERSION PIC X(1).
   03 VER-RELEASE PIC X(1).
   03 VER-MOD PIC X(1).
01 WORK-AREA.
   03 WORK-BIN PIC S9(4) BINARY.
   03 WORK-BIN-BY-BYTE REDEFINES WORK-BIN.
      05 WORK-BIN-BYTE1 PIC X(1).
      05 WORK-BIN-BYTE2 PIC X(1).
01 VERSION-F PIC 99.
01 RELEASE-F PIC 99.
01 MOD-F PIC 99.
01 PLATID PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
COPY CEEIGZCT.
   03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
   03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Serv-Lvl PIC X.
      03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.

Figure 130. COBOL Example of CEEGPID (Part 1 of 2)
PROCEDURE DIVISION.
PARA-CBLGPID:
  ** Call CEEGPID to return the version and
  ** platform ID
  CALL "CEEGPID" USING VERSION , PLATID , FC.
  IF NOT CEE000 of FC THEN
    DISPLAY "CEEGPID failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.

  ** If platform is OS/390 or VM and version is greater than 290,
  ** then use different format of VERSION.
  IF PLATID = 3 AND
  VERSION > 290 THEN
  ** Format the version, release, and modification level
  MOVE 0 to WORK-BIN
  MOVE VER-VERSION TO WORK-BIN-BYTE2
  MOVE WORK-BIN TO VERSION-F
  MOVE VER-RELEASE TO WORK-BIN-BYTE2
  MOVE WORK-BIN TO RELEASE-F
  MOVE VER-MOD TO WORK-BIN-BYTE2
  MOVE WORK-BIN TO MOD-F
  DISPLAY "Currently running version " VERSION-F
  " release " RELEASE-F " modification " MOD-F
  " of IBM Language Environment"
  ELSE
    DISPLAY "Currently running version " VERSION
    " of IBM Language Environment"
  END-IF

  ** Evaluate PLATID to display this platform
  EVALUATE PLATID
  WHEN 3
    DISPLAY "on OS/390 or VM"
  WHEN 4
    DISPLAY "on an AS/400"
  END-EVALUATE
GOBACK.

Figure 130. COBOL Example of CEEGPID (Part 2 of 2)

Figure 131 on page 321 is an example of CEEGPID called by PL/I.
CEEGQDT—Retrieve q_data_token

CEEGQDT retrieves the q_data_token from the Instance-Specific Information (ISI). CEEQDT is particularly useful when you have user-written condition handlers registered by CEEHDLR.

Syntax

```
CEEGQDT( cond_rep, q_data_token, fc )
```

cond_rep (input)

A condition token defining the condition for which the q_data_token is retrieved.
**CEEGQDT**

* q_data_token (output)
  A pointer to the q_data associated with condition token cond_rep.

* 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0EE</td>
<td>3</td>
<td>0462</td>
<td>Instance-specific information for the condition token with message number message-number and facility ID facility-id could not be found.</td>
</tr>
<tr>
<td>CEE0EG</td>
<td>3</td>
<td>0464</td>
<td>Instance-specific information for the condition token with message number message-number and facility ID facility-id did not exist.</td>
</tr>
</tbody>
</table>

**Usage notes**
- *z/OS UNIX consideration*—In multithread applications, CEEGQDT affects only the calling thread.

**For more information**
- For more information on the various types of q_data structures, see `Language Environment Programming Guide`.
- For more information about the CEEHDLR callable service, see “CEEHDLR—Register User-written condition handler” on page 336.
- For more information about the CEESGL callable service, see “CEESGL—Signal a condition” on page 453.

**Examples**
- Figure 132 on page 323 is an example of CEEGQDT called by C/C++.
/* Module/File Name: EDCGQDT */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>
#ifdef __cplusplus
extern "C" {
#endif
void handler(_FEEDBACK *, _INT4 *, _INT4 *, _FEEDBACK *);
#ifdef __cplusplus
}
#endif
typedef struct /* condition info structure */
{ int error_value;
  char err_msg[80];
  int retcode;
} info_struct;
int retcode;
info_struct *
int main(void)
{ /* register the condition handler */
  int 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;
}

Figure 132. C/C++ Example of CEEGQDT (Part 1 of 2)
CEEGQDT

/* 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",;
fcc.tok_msgno);
    exit(2999);
}
/* signal the condition */
CEESGL(&condtok,&qdata,&fc);
if (_FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEESGL failed with message number \d\n",;
fcc.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 */
    CEEQDT(fc, &idata, &qdatafc);
    if (_FBCHECK ( qdatafc , CEE000 ) != 0 ) {
        printf("CEEQDT failed with message number \d\n",;
            qdatafc.tok_msgno);
        *result = 20; /* percolate */
        return;
    }*/
    /* set info_struct pointer to address return by */
    /* CEEQDT */
    qdata = (info_struct *) idata;
    /* use the condition info structure (qdata) */
    if (qdata->error_value == 86) {
        printf("%12s", qdata->err_msg);
        printf("retcode = %d\n", qdata->retcode);
        *result = 10; /* resume this is what we want */
        return;
    }
    /* percolate */
    *result = 20; /* percolate */
}

Figure 132. C/C++ Example of CEEQDT (Part 2 of 2)

Figure 133 on page 325 is an example of CEEQDT called by COBOL.
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.
END-WORKING-STORAGE SECTION.
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**

```cobol
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**

```cobol
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 **

```cobol
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.
```

Figure 133. COBOL Example of CEEGQDT (Part 2 of 3)
Figure 134 on page 328 uses a COBOL program and handler to establish the condition handling environment prior to calling a PL/I subroutine to illustrate the use of the callable service from PL/I.
CBL LIB,QUOTE
*Module/File Name: IGZTGQDP
*************************************************
** IGZTGQDP - Drive sample program for CEEGQDT **
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGQDP.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 ROUTINE PROCEDURE-POINTER.
 01 TOKEN PIC S9(9) BINARY.
 01 SEV PIC S9(4) BINARY.
 01 MSGNO PIC S9(4) BINARY.
 01 CASE PIC S9(4) BINARY.
 01 SEV2 PIC S9(4) BINARY.
 01 CTRL 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.
    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.

Figure 134. PL/I Example of CEEQDQ (Part 1 of 2)
Figure 135 on page 330 is an example of CEEGQDT called by COBOL.
CEEGQDT

*PROCESS OPT(0), MACRO;
/* Module/File Name: IBMGQDT */
/***************************************************************************/
/**
/** Function: CEEGQDT -- get qualifying data
/**
/***************************************************************************/
IBMGQDT: PROC (@CONDTOK, @TOKEN, @RESULT, @NEWCOND)
    OPTIONS(COBOL);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

/* Parameters */
DCL @CONDTOK POINTER;
DCL @TOKEN POINTER;
DCL @RESULT POINTER;
DCL @NEWCOND POINTER;
DCL 01 CONDTOK BASED(@CONDTOK),
    / * Feedback token */
       03 MsgSev REAL FIXED BINARY(15,0),
       03 MsgNo REAL FIXED BINARY(15,0),
       03 Flags,
       05 Case BIT(2),
       05 Severity BIT(3),
       05 Control BIT(3),
       03 FacID CHAR(3), /* Facility ID */
       03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);
DCL TOKEN BASED(@TOKEN) REAL FIXED BIN(31,0);
DCL RESULT BASED(@RESULT) REAL FIXED BIN(31,0);
DCL 01 NEWCOND BASED(@NEWCOND),
    / * Feedback token */
       03 MsgSev REAL FIXED BINARY(15,0),
       03 MsgNo REAL FIXED BINARY(15,0),
       03 Flags,
       05 Case BIT(2),
       05 Severity BIT(3),
       05 Control BIT(3),
       03 FacID CHAR(3), /* Facility ID */
       03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);

Figure 135. COBOL Example of CEEGQDT (Part 1 of 2)
CEEGTST—Get heap storage

CEEGTST gets storage from a heap whose ID you specify. It is used to acquire both large and small blocks of storage. CEEGTST always allocates storage that is addressable by the caller. Therefore, if the caller is in 24-bit addressing mode, or if HEAP(,BELOW) is in effect, the storage returned is always below the 16M line. Above-the-line storage is returned only if the caller is in 31-bit addressing mode and HEAP(,ANY) is in effect.

All requests for storage are conditional. If storage is not available, the feedback code (fc) is set and returned to you, but the thread does not abend. When storage is not available, the appropriate action in the member environment should be taken. One option is to use the CEESGL callable service to signal the Language Environment condition handler with the returned feedback code.

Storage obtained by CEEGTST can be freed by a call to CEEFRST or CEEDSHP. You can also free storage by using a language intrinsic function. If storage is not explicitly freed, it is freed automatically at termination.

If you have specified a heap_alloc_value in the STORAGE 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

```

├── CEEGTST(─heap_id─,─size─,─address─,─fc─)
```

heap_id (input)

A fullword binary signed integer. heap_id is a token denoting the heap in which the storage is allocated. A heap_id of 0 allocates storage from the initial heap (or user heap). Any other heap_id must be a value obtained from the CEECRHP callable service. If the heap_id you specify is invalid, no storage is allocated. CEEGTST terminates with a non-CEE000 symbolic feedback code and the value of the address parameter is undefined.

size (input)

A fullword binary signed integer. size represents the amount of storage allocated, in bytes. If the specified amount of storage cannot be obtained, no storage is allocated, CEEGTST terminates with a non-CEE000 symbolic feedback code, and the value of the address parameter is undefined.

address (output)

A fullword address pointer. address is the main storage address of the first byte of allocated storage. If storage cannot be obtained, address remains undefined. Storage is always allocated on a doubleword boundary. This parameter contains an address that is returned as output to CEECZST.

fc (output)

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Codes can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0P2</td>
<td>4</td>
<td>0802</td>
<td>Heap storage control information was damaged.</td>
</tr>
<tr>
<td>CEE0P3</td>
<td>3</td>
<td>0803</td>
<td>The heap identifier in a get storage request or a discard heap request was unrecognized.</td>
</tr>
<tr>
<td>CEE0P8</td>
<td>3</td>
<td>0808</td>
<td>Storage size in a get storage request (CEEGTST) or a reallocate request (CEECZST) was not a positive number.</td>
</tr>
<tr>
<td>CEE0PD</td>
<td>3</td>
<td>0813</td>
<td>Insufficient storage was available to satisfy a get storage (CEECZST) request.</td>
</tr>
</tbody>
</table>


Usage notes

- COBOL considerations—If you want to use the CEEGTST callable service in a 24-bit addressing mode COBOL program to request 24-bit heap storage, you can make a static call to CEEGTST without any additional changes. If you want to call it dynamically (either by using the COBOL DYNAM compiler option, or using the "CALL identifier" statement, where identifier is a variable that holds the name of the program you want to call), you must also specify HEAPP(BELOW) as a run-time option.

This run-time specification affects all programs in the enclave using the user heap, not just the 24-bit addressing mode program. If this is undesirable, one alternative is to use the CEECRHP callable service to create an additional heap, specifying for the options parameter a value that will create the heap BELOW. The 24-bit addressing mode COBOL program can then obtain storage from this additional heap using the CEEGTST callable service.

- PL/I considerations—Storage allocated within PL/I AREAs is managed by PL/I. Therefore, only PL/I language functions can allocate and free storage within a PL/I area.

- Based upon the layout of a PL/I structure, PL/I might adjust the starting byte of the PL/I structure to a non-doubleword aligned byte. The difference between the doubleword boundary and the first byte of such a structure is known as the hang. Because Language Environment callable storage services do not adjust the starting byte, you must be careful using callable services to allocate storage for PL/I structures. Use either the PL/I ALLOCATE statement or fully defined structures and aggregates to avoid this problem.

- CICS considerations—in a CICS environment, size should not exceed 1024M (1 gigabyte or X'40000000') when running in AMODE ANY, and 65,504 bytes when running in 24-bit addressing mode. These CICS restrictions are subject to change from one release of CICS to another. Portable applications should respect current CICS limitations.

- z/OS UNIX considerations—CEEGTST applies to the enclave. Any thread can free the allocated storage.

For more information

- See "HEAP" on page 36 for further information about the Language Environment HEAP run-time option.
- For more information about the CEESGL callable service, see "CEESGL—Signal a condition" on page 453.
- For more information about the CEEFRST callable service, see "CEEFRST—Free heap storage" on page 298.
- For more information about the CEEDSHP callable service, see "CEEDSHP—Discard heap" on page 268.
- For more information about the STORAGE run-time option, see "STORAGE" on page 75.
- For more information about the CEECRHP callable service, see "CEECRHP—Create new additional heap" on page 227.
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 68.
- For more information about the CEECRHP callable service, see "CEECRHP—Create new additional heap" on page 227.
- For more information about the CEECRHP callable service, see "CEECRHP—Create new additional heap" on page 227.
- For more information about the CEECZST callable service, see "CEECZST—Reallocate (change size of) storage" on page 232.
Examples

Figure 136 is an example of CEEGTST called by C/C++.

```c
/*Module/File Name: EDCGTST */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>
int main(void) {
    _INT4 heapid, size;
    _POINTER address;
    _FEEDBACK fc;
    /* .
    .*/
    heapid = 0; /* get storage from initial heap */
    size = 4000; /* number of bytes of heap storage */
    /* obtain the storage using CEEGTST */
    CEEGTST(&heapid,&size,&address,&fc);
    /* check the first 4 bytes of the feedback token */
    /* (0 if successful) */
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEGTST failed with message number %d\n",
               fc.tok_msgno);
        exit(99);
    }
    /* .
    .*/
    /* free the storage that was previously obtained */
    /* using CEEGTST */
    CEEFRST(&address,&fc);
    /* check the first 4 bytes of the feedback token */
    /* (0 if successful) */
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEFRST failed with message number %d\n",
               fc.tok_msgno);
        exit(99);
    }
    /* .
    .*/
    }
```

Figure 136. C/C++ Example of CEEGTST

Figure 137 on page 335 is an example of CEEGTST called by COBOL.
**Figure 137. COBOL Example of CEEGTST**

*Figure 138 on page 336* is an example of CEEGTST called by PL/I.
## CEEHDLR—Register User-written condition handler

CEEHDLR registers a user-written condition handler for the current stack frame. The user condition handler is invoked when:

- It is registered for the current stack frame by CEEHDLR, and
- The Language Environment condition manager requests the condition handler associated with the current stack frame handle the condition.

Language Environment places the user-written condition handlers associated with each stack frame in a queue. The queue can be empty at any given time. The Language Environment condition manager invokes the registered condition handlers in LIFO (last in, first out) order to handle the condition.

The opposite of CEEHDLR, which registers a user-written condition handler, is CEEHDLU, which unregisters the handler. You do not necessarily need to use CEEHDLU to remove user-written condition handlers you registered with CEEHDLR. Any user-written condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by Language Environment, but only when the associated stack frame is removed from the stack.

---

### Figure 138. PL/I Example

```pli
*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 CEEIBMWW;
%INCLUDE CEEIBMCT;

DCL HEAPID REAL FIXED BINARY(31,0);
DCL STGSIZE REAL FIXED BINARY(31,0);
DCL ADDRSS POINTER;
DCL 01 FC, /* Feedback token */
         03 MsgSev REAL FIXED BINARY(15,0),
         03 MsgNo REAL FIXED BINARY(15,0),
         03 Flags,
         05 Case BIT(2),
         05 Severity BIT(3),
         05 Control BIT(3),
         03 FacID CHAR(3), /* Facility ID */
         03 ISI /* Instance-Specific Information */
         REAL FIXED BINARY(31,0);

HEAPID = 0; /* get storage from the initial heap */
STGSIZE = 4000; /* get 4000 bytes of storage */

/* Call CEEGTST to obtain the storage */
CALL CEEGTST ( HEAPID, STGSIZE, ADDRSS, FC );

   IF FBCHECK( FC, CEE000) THEN DO;
      PUT SKIP LIST( 'Obtained ' || STGSIZE
                     || ' bytes of storage at location
                     ' || DECIMAL( UNSPEC( ADDRSS1 ) )
                     || ' from heap ' || HEAPID );
   END;
   ELSE DO;
      DISPLAY( 'CEEGTST failed with msg'
               || FC.MsgNo );
      STOP;
   END;

END PLIGTST;
```
Language Environment condition handlers are driven only for synchronous conditions.

Syntax

```
CEEHDLR((routine, token, fc))
```

**routine (input)**

An entry variable or entry constant for the routine called to process the condition. The entry variable or constant must be passed by reference. The routine must be an external routine; that is, it must not be a nested routine.

**token (input)**

A fullword integer of information you want passed to your user handler each time it is called. This can be a pointer or any other fullword integer you want to pass.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE080</td>
<td>1</td>
<td>0256</td>
<td>The user-written condition handler routine specified was already registered for this stack frame. It was registered again.</td>
</tr>
<tr>
<td>CEE081</td>
<td>3</td>
<td>0257</td>
<td>The routine specified contained an invalid entry variable.</td>
</tr>
</tbody>
</table>

**Usage notes**

- PL/I MTF consideration—CEEHDLR is not supported in PL/I MTF applications. This includes any CEEHDLR service called from a COBOL program in the application.
- COBOL consideration—You should not call CEEHDLR from a nested COBOL program.
- z/OS UNIX consideration—In multithread applications, CEEHDLR affects only the calling thread.
- C consideration—The CEEHDLR service does not save Writeable Static Area (WSA) information about the user handler, so it's possible that the user handler will be given control with the wrong WSA. Specifically, the user handler(s) on the stack will be invoked with the WSA of the routine that incurred the condition. The preferred method of handling conditions from a C application is to use C signal handlers.

**For more information**

- For more information about the CEEHDLU callable service, see “CEEHDLU—Unregister user-written condition handler” on page 346.

**Examples**

Figure 139 on page 338 is an example of CEEHDLR called by C/C++. 
Figure 139. C/C++ Example of CEEHDLR

Figure 140 on page 339 is an example of CEEHDLR called by a COBOL program that registers a handler routine.
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLHDLR.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE PROCEDURE-POINTER.
    01 TOKEN PIC S9(9) 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 CONDOK.
        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.

Figure 140. COBOL Program that Registers Handler Routine (Part 1 of 2)
* RAISE A SIGNAL

PARA-CBLSGL.

******************************************************************************
** Call CEENCOD with the values assigned above **
** to build a condition token "CONDTOK" **
** Set CONDTOK to sev=3, msgno=1 facid=CEE. We **
** raise a sev 3 to ensure our handler is driven*
******************************************************************************
MOVE 3 TO SEV.
MOVE 1 TO MSGNO.
MOVE 1 TO CASE.
MOVE 3 TO SEV2.
MOVE 1 TO CNTRL.
MOVE "CEE" TO FACID.
MOVE 0 TO ISINFO.

CALL "CEENCOD" USING SEV, MSGNO, CASE,
SEV2, CNTRL, FACID, ISINFO, CONDTOK, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEENCOD failed with msg "
   Msg-No of FC UPON CONSOLE
   STOP RUN
END-IF.

******************************************************************************
** Call CEESGL to signal the condition with **
** the condition token and qdata described **
** in CONDTOK and QDATA **
******************************************************************************
MOVE 0 TO QDATA.
CALL "CEESGL" USING CONDTOK, QDATA, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEESGL failed with msg "
   Msg-No of FC UPON CONSOLE
   STOP RUN
END-IF.

GOBACK.

Figure 140. COBOL Program that Registers Handler Routine (Part 2 of 2)

Figure 141 on page 341 is an example of a COBOL user-written condition handler
that is registered by CBLHDLR and unregistered by CBLHDLU.
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.

Figure 141. COBOL User-Written Condition Handler Registered by CBLHDLR and Unregistered by CBLHDLU (Part 1 of 2)
Figure 142 on page 343 is an example of a PL/I program to handle divide-by-zero condition.
Figure 142. EXCOND Program (PL/I) to Handle Divide-by-Zero Condition (Part 1 of 2)
/*********************************************/
/* Unregister the user condition handler */
/*********************************************/

Call ceehdlu (Usrhdlr, fback);
If fbcheck (fback, cee000) then
display ('MAIN: unregistered USRHDLR');
else
do;
display ('CEEHDLU failed with message number ' ||
    fback.MsgNo);
    stop;
end;

/*********************************************/
/* Subroutine that simply raises ZERO DIVIDE */
/*********************************************/
divzero: proc (arg);
dcl arg fixed bin(31);
display(' DIVZERO: starting. ');
    arg = 1 / arg;
display(' DIVZERO: Returning to its caller');
end divzero;
end Excond;

Figure 142. EXCOND Program (PL/I) to Handle Divide-by-Zero Condition (Part 2 of 2)
AHDL TITLE 'Main program that registers a handler'
*
* Symbolic Register Definitions and Usage
*
R0 EQU 0 Parm list addr (CMS only)
R1 EQU 1 Parm list addr, 0=no parms
R10 EQU 10 Base reg for executable code
R12 EQU 12 Language Environment Common Anchor Area addr
R13 EQU 13 Dynamic Storage Area addr
R14 EQU 14 Return point addr
R15 EQU 15 Entry point address
*
* Prologue
*
CEEHDLRA CEEENTRY AUTO=OSASIZ, Main memory to obtain *
MAIN=YES, This program is a MAIN prog *
PPA=PPA1, Our Program Prolog Area *
BASE=R10 Base reg for executable code
USING CEECAA,R12 Addressing for LE/370 CAA
USING CEEDSA,R13 Addressing for dynamic data
*
* Announce ourselves
*
WTO 'CEEHDLRA Says "HELLO"',ROUTCDE=11
*
* Register User Handler
*
LA R1,USRHDLPP Get addr of proc-ptr to Hdlr
ST R1,PARM1 Make it 1st parameter
LA R1,TOKEN Get addr of 32-bit token
ST R1,PARM2 Make it 2nd parameter
LA R1,FEEDBACK Get addr of Feedback Code
ST R1,PARM3 Make it 3rd parameter
LA R1,HDLRPLST Point to CEEHDLR's parm list
CALL CEEHDLR Invoke CEEHDLR service
CLC FEEDBACK,=XL12'00' Check for success..
BE HDLRGOOD Skip diagnostics if success
*
WTO '**** Call to CEEHDLR failed ****', *
ROUTCDE=11
*
ABEND 1,DUMP Terminate program with Dump
HLRGOOD EQU * Handler registered OK

Figure 143. PL/I Example of CEEHDLR (Part 1 of 2)
CEEHDLU

CEEHDLU—Unregister user-written condition handler

CEEHDLU unregisters a user condition handler for the current stack frame. You do not necessarily need to use CEEHDLU to remove user-written condition handlers you registered with CEEHDLR. Any user-written condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by Language Environment, but only when the associated stack frame is removed from the stack.

Note: For information about restrictions on the use of CEEHDLU with PL/I, see z/OS Language Environment Programming Guide.
CEEHDLU

Syntax

```
>> CEEHDLU(—routine—,—fc—)
```

**routine (input)**

An entry variable or constant for the routine to be unregistered as a user condition handler. This routine must be previously registered (with CEEHDLR) by the same stack frame that invokes CEEHDLU.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE07S</td>
<td>1</td>
<td>0252</td>
<td>CEEHDLU was unable to find the requested user-written condition handler routine.</td>
</tr>
</tbody>
</table>

**Usage notes**

- PL/I MTF consideration—CEEHDLU is not supported in PL/I MTF applications. This includes any CEEHDLR service called from a COBOL program in the application.
- z/OS UNIX consideration—In multithread applications, CEEHDLU affects only the calling thread.

**For more information**

- See "CEEHDLR—Register User-written condition handler" on page 336 for further information about specifying the `routine` parameter.

**Examples**

Figure 144 on page 348 is an example of CEEHDLU called by C/C++. 
Figure 144. C/C++ Example of CEEHDLU

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result, 
               _FEEDBACK *newfc) {
    /*...
    */
}
** Figure 146 on page 350 is an example of a COBOL user-written condition handler that is registered by CBLHDLR and unregistered by CBLHDLU. **
**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.

Figure 146. COBOL User-Written Condition Handler Registered by CBLHDLR and
Unregistered by CBLHDLU (Part 2 of 2)

**CEEISEC—Convert integers to seconds**

CEEISEC converts separate binary integers representing year, month, day, hour,
minute, second, and millisecond to a number representing the number of seconds
since 00:00:00 14 October 1582. Use CEEISEC instead of CEESECS when the input
is in numeric format rather than character format.
The inverse of CEEISEC is CEESECI, which converts number of seconds to integer year, month, day, hour, minute, second, and millisecond.

**Syntax**

```
CEEISEC((input_year, input_months, input_day, input_hours, input_minutes, input_seconds, input_milliseconds), (output_seconds, fc))
```

**input_year (input)**
A 32-bit binary integer representing the year. The valid range for input_year is 1582 to 9999, inclusive.

**input_month (input)**
A 32-bit binary integer representing the month. The valid range for input_month is 1 to 12.

**input_day (input)**
A 32-bit binary integer representing the day. The valid range for input_day is 1 to 31.

**input_hours (input)**
A 32-bit binary integer representing the hours. The range of valid input_hours is 0 to 23.

**input_minutes (input)**
A 32-bit binary integer representing the minutes. The range of valid input_minutes is 0 to 59.

**input_seconds (input)**
A 32-bit binary integer representing the seconds. The range of valid input_seconds is 0 to 59.

**input_milliseconds (input)**
A 32-bit binary integer representing milliseconds. The range of valid input_milliseconds is 0 to 999.

**output_seconds (output)**
A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range of output_seconds is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999). If any input values are invalid, output_seconds is set to zero. To convert output_seconds to a Lilian day number, divide output_seconds by 86,400 (the number of seconds in a day).

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2EE</td>
<td>3</td>
<td>2510</td>
<td>The hours value in a call to CEEISEC or CEESECS was not recognized.</td>
</tr>
<tr>
<td>CEE2EF</td>
<td>3</td>
<td>2511</td>
<td>The day parameter passed in a CEEISEC call was invalid for year and month specified.</td>
</tr>
</tbody>
</table>
### CEEISEC

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE2EH</td>
<td>3</td>
<td>2513</td>
<td>The input date passed in a CEEISEC, CEEDAYS, or CEESECS call was not within the supported range.</td>
</tr>
<tr>
<td>CEE2EI</td>
<td>3</td>
<td>2514</td>
<td>The year value passed in a CEEISEC call was not within the supported range.</td>
</tr>
<tr>
<td>CEE2EJ</td>
<td>3</td>
<td>2515</td>
<td>The milliseconds value in a CEEISEC call was not recognized.</td>
</tr>
<tr>
<td>CEE2EK</td>
<td>3</td>
<td>2516</td>
<td>The minutes value in a CEEISEC call was not recognized.</td>
</tr>
<tr>
<td>CEE2EL</td>
<td>3</td>
<td>2517</td>
<td>The month value in a CEEISEC call was not recognized.</td>
</tr>
<tr>
<td>CEE2EN</td>
<td>3</td>
<td>2519</td>
<td>The seconds value in a CEEISEC call was not recognized.</td>
</tr>
</tbody>
</table>

**Usage notes**

- z/OS UNIX consideration—In multithread applications, CEEISEC affects only the calling thread.

**For more information**

- For more information about the CEESECS callable service, see "CEESECS—Convert timestamp to seconds" on page 441.
- See "CEESECI—Convert seconds to integers" on page 436 for more information about the CEESECI callable service.

**Examples**

Figure 147 on page 354 is an example of CEEISEC called by C/C++.
#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);
}

Figure 147. C/C++ Example of CEEISEC

Figure 148 on page 355 is an example of CEEISEC called by COBOL.
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.

Figure 148. COBOL Example of CEEISEC (Part 1 of 2)
Figure 148. COBOL Example of CEEISEC (Part 2 of 2)

Figure 149 is an example of CEEISEC called by PL/I.

**************
** 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.

Figure 149. PL/I Example of CEEISEC (Part 1 of 2)

+PROCESS MACRO;
/*Module/File Name: IBMISEC
/*****************************/
/**
/** Function: CEEISEC - Convert integers to
/** seconds
/**
/** In this example, CEEISEC is called to convert
/** integers representing the date and time to the
/** number of seconds since 00:00 14 October 1582.
/**
/*****************************/
PLIISEC: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;
DCL YEAR REAL FIXED BINARY(31,0);
DCL MONTH REAL FIXED BINARY(31,0);
DCL DAYS REAL FIXED BINARY(31,0);
DCL HOURS REAL FIXED BINARY(31,0);
DCL MINUTES REAL FIXED BINARY(31,0);
DCL SECONDS REAL FIXED BINARY(31,0);
DCL MILLSEC REAL FIXED BINARY(31,0);
DCL OUTSECS REAL FLOAT DECIMAL(16);
DCL 01 FC, /* Feedback token */
   03 Flags,
   05 Case BIT(2),
   05 Severity BIT(3),
   05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);
/* Specify integers representing */
/* 00:00:00 1 January 2000 */
YEAR = 2000;
MONTH = 1;
DAYS = 1;
HOURS = 0;
MINUTES = 0;
SECONDS = 0;
MILLSEC = 0;
CEEITOK

/* 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' );
ENDIF;
ELSE DO;
    DISPLAY( 'CEEISEC failed with msg ' || FC.MsgNo );
    STOP;
ENDIF;
END PLIISEC;

Figure 149. PL/I Example of CEEISEC (Part 2 of 2)

**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

```plaintext
>>> CEEITOK( i_ctok, fc ) <<<
```

**i_ctok (output)**

A 12-byte condition token identifying the initial condition in the current active data block being processed.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE35S</td>
<td>1</td>
<td>3260</td>
<td>No condition was active when a call to a condition management routine was made.</td>
</tr>
</tbody>
</table>

### Usage notes

- **z/OS UNIX considerations**—In multithread applications, CEEITOK affects only the calling thread. CEEITOK returns the initial token for the condition of the thread.

### For more information

- See "CEENCOD—Construct a condition token" on page 407 for more information about the CEENCOD callable service.

### Examples

Figure 150 on page 358 is an example of CEEITOKC called by C/C++.
C/EITOK

/*@Module/File Name: EDCITOK */

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

#ifdef __cplusplus
extern "C" {
#endif

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

#ifdef __cplusplus
}
#endif

int main(void) {
    _FEEDBACK fc,condtok;
    _ENTRY routine;
    _INT4 token,qdata;
    _INT2 c_1,c_2,cond_case,sev,control;
    _CHAR3 facid;
    _INT4 isi;

    /* register condition handler */
    token = 99;
    routine.address = (_POINTER)&handler;
    routine.nesting = NULL;
    CEEHDLR(&routine,&token,&fc);
    if ( _FBCHECK ( fc , CEE000 ) !=0 ) {
        printf("CEEHDLR failed with message number %d\n",
                fc.tok_msgno);
        exit(2999);
    }

    /* build the condition token */
    c_1 = 1;
    c_2 = 99;
    cond_case = 1;
    sev = 1;
    control = 0;
    memcpy(facid,"ZZZ",3);
    isi = 0;
}

Figure 150. C/C++ Example of CEEITOK (Part 1 of 2)
Figure 150. C/C++ Example of CEEITOK (Part 2 of 2)

Figure 151 on page 360 is an example of CEEITOKC called by COBOL.
CBL LIB,QUOTE,NOOPT

*Module/File Name: IGZITOK

*************************************************
** 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.

Figure 151. COBOL Example of CEEITOK (Part 1 of 3)
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.

Figure 151. COBOL Example of CEEITOK (Part 2 of 3)
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
  END-IF.
PARA-HANDLER.

******************************************************************************
** In user handler - resume execution **
******************************************************************************
SET RESUME TO TRUE.
GOBACK.

END PROGRAM CBLITOK.

Figure 151. COBOL Example of CEEITOK (Part 3 of 3)
**CEELCNV—Query Locale numeric conventions**

CEELCNV, which calls the C function `localeconv()`, queries the numeric formatting information from the current locale and sets the components of a structure with values pertaining to the LC_NUMERIC and LC_MONETARY categories. It sets the components of an object of the type NM_STRUCT with the values appropriate for the formatting of the numeric quantities (monetary and otherwise) according to the rules of the current locale.

CEELCNV is sensitive to the locales set by `setlocale()` or `CEESETL`, not to the Language Environment settings from COUNTRY or CEE3CTY.
CEELCNV

Syntax

CEELCNV(\textasciitilde version\textasciitilde,\textasciitilde num\_and\_mon\textasciitilde,\textasciitilde fc\textasciitilde)

version

This parameter points to a user supplied VERSION_INFO structure, in which the first four bytes contain the callable service version number.

Requirement: If the parameter does not points to a user supplied VERSION_INFO structure, it must be omitted. If the parameter points to a VERSION_INFO containing version number 2, CEELCNV returns a num\_and\_mon structure with the international monetary string formatting elements added for ISO/IEC 9899:1999 (C99). This corresponds to the current C language lconv structure. If the parameter is omitted, CEELCNV returns the structure format used before C99. For information on how to code this parameter, see "Invoking callable services" on page 112.

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:

\textbf{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:

\textbf{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.

\textbf{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.

\textbf{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.

\textbf{VSTRING int\_curr\_symbol}

A halfword length-prefixed character string (VSTRING) of 22 bytes that is the international currency symbol applicable to the current locale, left justified within a four-character space-padded field.
VSTRING currency_symbol
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the local currency symbol applicable to the current locale.

VSTRING mon_decimal_point
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the decimal point character used to format monetary quantities.

VSTRING mon_thousands_sep
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the separator for groups of digits to the left of the decimal point in formatted monetary quantities.

VSTRING mon_grouping
A halfword length-prefixed character string (VSTRING) of 22 bytes whose elements indicate the size of each group of digits in formatted monetary quantities.

VSTRING positive_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes that indicates a non-negative-formatted monetary quantity.

VSTRING negative_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes used to indicate a negative-formatted monetary quantity.

VINT int_frac_digits
A 1-byte integer that is the number of fractional digits (those to the right of the decimal point) to be displayed in an internationally-formatted monetary quantity.

VINT frac_digits
A 1-byte integer that is the number of fractional digits (those to the right of the decimal point) to be displayed in a formatted monetary quantity.

VINT p_cs_precedes
A 1-byte integer that is set to 1 if the currency_symbol precedes the value for a non-negative-formatted monetary quantity. It is set to 0 if it follows.

VINT p_sep_by_space
A 1-byte integer that is set to 1 if the currency_symbol is separated by a space from the value for a non-negative-formatted monetary quantity. It is set to 0 if it is not separated.

VINT n_cs_precedes
A 1-byte integer that is set to 1 if the currency_symbol precedes the value for a negative-formatted monetary quantity. It is set to 0 if it follows.

VINT n_sep_by_space
A 1-byte integer that is set to 1 if the currency_symbol is separated by a space from the value for a negative-formatted monetary quantity. It is set to 0 if it is not separated.

VINT p_sign_posn
A 1-byte integer that is set to a value indicating the positioning of the positive_sign for non-negative-formatted monetary quantity.
VINT n_sign_posn
A 1-byte integer that is set to a value indicating the position of the
negative_sign for negative-formatted monetary quantity.

VSTRING left_parenthesis
A halfword length-prefixed character string (VSTRING) of 22 bytes that
is the used to denote a negative monetary quantity.

VSTRING right_parenthesis
A halfword length-prefixed character string (VSTRING) of 22 bytes that
is the used to denote a negative monetary quantity.

VSTRING debit_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes that
is the debit sign character in monetary formats.

VSTRING credit_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes that
is the credit sign character in monetary formats.

VINT int_p_cs_precedes
A 1-byte integer that is set to 1 if the currency_symbol precedes the
value for a non-negative-formatted international monetary quantity. It
is set to 0 if it follows.

VINT int_p_sep_by_space
A 1-byte integer that is set to 1 if the currency_symbol is separated by a
space from the value for a non-negative-formatted international
monetary quantity. It is set to 0 if it is not separated.

VINT int_n_cs_precedes
A 1-byte integer that is set to 1 if the currency_symbol precedes the
value for a negative-formatted international monetary quantity. It is set
to 0 if it follows.

VINT int_n_sep_by_space
A 1-byte integer that is set to 1 if the currency_symbol is separated by a
space from the value for a negative-formatted international monetary
quantity. It is set to 0 is it is not separated.

VINT int_p_sign_posn
A 1-byte integer that is set to a value indicating the positioning of the
positive_sign for non-negative-formatted international monetary
quantity.

VINT n_sign_posn
A 1-byte integer that is set to a value indicating the position of the
negative_sign for negative-formatted international monetary quantity.

fc (output/optional)
A 12-byte feedback code, optional in some languages, that indicates the result
of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
<tr>
<td>CEE3VN</td>
<td>3</td>
<td>4087</td>
<td>Input Error: The version_info control block contains a version number that is not valid.</td>
</tr>
</tbody>
</table>
Usage notes

- PL/I MTF consideration—CEELCNV is not supported in PL/I MTF applications.
- This callable service uses the C/C++ run-time library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.
- CEELCNV does not return all numeric conventions.
- CHAR_MAX determines when no further grouping is to be performed. The elements of grouping and mon_grouping are interpreted according to the following CHAR_MAX settings:
  - 0 specifies that the previous element is to be repeatedly used for the remainder of the digits. Indicates that no further grouping is to be performed.
  - Any other value represents the number of digits comprising the current group. The next element is examined to determine the size of the next group of digits to the left of the current group.
- The value of p_sign_posn and n_sign_posn is interpreted according to the following:
  0 Parentheses surround the quantity and currency_symbol.
  1 The sign string precedes the quantity and currency_symbol.
  2 The sign string follows the quantity and currency_symbol.
  3 The sign string immediately precedes the currency_symbol.
  4 The sign string immediately follows the currency_symbol.

For more information

- See “CEEQRYL—Query active locale environment” on page 423 for a description of LC_NUMERIC and LC_MONETARY categories.
- See “CEESETL—Set locale operating environment” on page 448 for details on the CEESETL callable service.

Examples

Figure 153 on page 368 is an example of CEELCNV called by COBOL.
** Example for callable service CEELCNV  
** Function: Retrieve numeric and monetary  
** format for default locale and  
** print an item.  
** Set locale to France, retrieve  
** structure, and print an item.  
** Valid only for COBOL for MVS & VM Release 2  
** or later.  
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINLCNV.
DATA DIVISION.
WORKING-STORAGE SECTION.
*************************************************
** Use Locale NM-Struct for CEELCNV calls  
*************************************************
COPY CEEIGZN2M.
*
PROCEDURE DIVISION.
*************************************************
** Subroutine needed for addressing  
*************************************************
** Subroutine needed for addressing  
CALL "COBLCNV" USING NM-Struct.
STOP RUN.
*
IDENTIFICATION DIVISION.
PROGRAM-ID. COBLCNV.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PTR1 Pointer.
01 Locale-Name.
   02 LN-Length PIC S9(54) BINARY.
   02 LN-String PIC X(256).
*************************************************
** Use Locale category constants  
*************************************************
COPY CEEIGZLC.
*
01 FC.
   02 Condition-Token-Value.
COPY CEEIGZCT.
   03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
   03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
   03 Case-Sev-Ctl
      02 I-S-Info PIC S9(9) BINARY.
      03 Facility-ID PIC XXX.
      03 Facility-ID PIC XXX.
      02 1-S-Info PIC S9(9) BINARY.

Figure 153. COBOL Example of CEELCNV (Part 1 of 2)
Figure 153. COBOL Example of CEELCNV (Part 2 of 2)

Figure 154 on page 370 is an example of CEELCNV called by PL/I.
**CEELOCT**

CEELOCT—Get current local date or time

CEELOCT returns the current local date or time in three formats:

- Lilian date (the number of days since 14 October 1582)
- Lilian seconds (the number of seconds since 00:00:00 14 October 1582)
- Gregorian character string (in the form YYYYMMDDHHMISS999)

These values are compatible with other Language Environment date and time services, and with existing language intrinsic functions.

---

**Figure 154. PL/I Example of CEELCNV**

```pli
*PROCESS MACRO;
/*Module/File Name: IBMCLCNV */
/***************************************************************************/
/* Example for callable service CEELCNV */
 /***************************************************************************/
/* Function: Retrieve numeric and monetary format */
/* structure for default locale and print an item. */
/* Set locale to France, retrieve structure and */
/* print an item. */
/***************************************************************************/
PLILCNV: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs */
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants */
%INCLUDE CEEIBMLC; /* Locale category constants */
%INCLUDE CEEIBMN2M; /* NM_STRUCT for CEELCNV calls */
/* use explicit pointer for local NM_STRUCT struct */
DCL NUM_AND_MON POINTER INIT(ADDR(NM_STRUCT));
/* Point to local version_info struct and initialize*/
/* VERSION TO 2 TO USE C99 MAPPING OF NM_STRUCT */
DCL VERSN POINTER INIT(ADDR(version_info));
VERSION_INFO.VERSION = 2;
/* CEESETL service call arguments */
DCL LOCALE_NAME CHAR(256) VARYING;
DCL 01 FC, /* Feedback token */
 03 MsgSev REAL FIXED BINARY(15,0),
 03 MsgNo REAL FIXED BINARY(15,0),
 03 Flags,
 05 Case BIT(2),
 05 Severity BIT(3),
 05 Control BIT(3),
 03 FacID CHAR(3), /* Facility ID */
 03 ISI /* Instance-Specific Information */
REAL FIXED BINARY(31,0);
/* retrieve structure for default locale */
CALL CEELCNV ( *, NUM_AND_MON, FC );
PUT SKIP LIST('Default DECIMAL_POINT is ',
 NM_STRUCT.DECIMAL_POINT);
/* set locale for France */
LOCALE_NAME = 'Fr_FRFRAN';
/* use LC_NUMERIC category const from CEEIBMLC */
CALL CEESETL ( LOCALE_NAME, LC_NUMERIC, FC );
/* use LC_MONETARY category const from CEEIBMLC */
CALL CEESETL ( LOCALE_NAME, LC_MONETARY, FC );
/* FBCHECK macro used (defined in CEEIBMCT) */
IF FBCHECK( FC, CEE000 ) THEN DO;
/* retrieve active NM_STRUCT, France Locale */
CALL CEELCNV ( *VERSN, NUM_AND_MON, FC );
PUT SKIP LIST('French DECIMAL_POINT is ',
 NM_STRUCT.DECIMAL_POINT);
END;
END PLILCNV;
```
CEEOCT 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

```c
CEEOCT(output_Lilian, output_seconds, output_Gregorian, fc)
```

**output_Lilian (output)**
A 32-bit binary integer representing the current local date in the Lilian format, that is, day 1 equals 15 October 1582, day 148,887 equals 4 June 1990. If the local time is not available from the system, output_Lilian is set to 0 and CEEOCT 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 CEEOCT terminates with a non-CEE000 symbolic feedback code.

**output_Gregorian (output)**
A 17-byte fixed-length character string in the form YYYYMMDDHHMISS9999 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 Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2F3</td>
<td>3</td>
<td>2531</td>
<td>The local time was not available from the system.</td>
</tr>
</tbody>
</table>

Usage notes

• z/OS consideration—The MVS command SET DATE will not affect the value returned by CEEOCT.
• CICS consideration—CEEOCT does not use the OS TIME macro.
• z/OS UNIX consideration—in multithread applications, CEEOCT affects only the calling thread.
CEELOCT

For more information

- See "CEEDATM—Convert seconds to character timestamp" on page 243 for more information about the CEEDATM callable service.

Examples

Figure 155 is an example of CEELOCT called by C/C++.

```c
/*Module/FileName: 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);
}
```

Figure 155. C/C++ Example of CEELOCT

Figure 156 on page 373 is an example of CEELOCT called by COBOL.
Figure 157 on page 374 is an example of CEELOCT called by PL/I.
CEEMGET—Get a message

CEEMGET retrieves, formats, and stores in a passed message area a message corresponding to a condition token that is either returned from a callable service or passed to a user-written condition handler. The caller can later retrieve the message to change or to write as output.

Syntax

```
>>CEEMGET((cond_token, message_area, msg_ptr, fc))
```

cond_token (input/output)
A 12-byte condition token received or returned as the result of a Language Environment callable service.

**Figure 157. PL/I Example of CEELCT**

```pli
+PROCESS MACRO;
/** Module/File Name: IBMLOCT */
/*****************************/
/**
/** Function: CEELCT - get current local time
/**
/** In this example, CEELCT is called to return
/** the current local time as a Lillian date,
/** Lillian timestamp, and Gregorian character
/** string.
/**
/*****************************************************************************/
PLILOCT: PROC OPTIONS(MAIN);
INCLUDE CEEIBMAW;
INCLUDE CEEIBMCT;
DCL LILLIAN REAL FIXED BINARY(31,0);
DCL SECONDS REAL FLOAT DECIMAL(16);
DCL GREGORN CHARACTER (17);
DCL 01 FC, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);
/* Call CEELCT to return local time in 3 formats */
CALL CEELCT (LILLIAN, SECONDS, GREGORN, FC);
/* If CEELCT 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( 'CEELCT failed with msg ' || FC.MsgNo );
  STOP;
  END;
END PLILOCT;
```

```
CEEMGET

**message_area** *(input/output)*
A fixed-length 80-character string (VSTRING), where the message is placed. The message is left-justified and padded on the right with blanks.

**msg_ptr** *(input/output)*
A 4-byte binary integer returned to the calling routine. The `msg_ptr` should be passed a value of zero on the initial call to CEEMGET. If a message is too large to be contained in the `message_area`, `msg_ptr` (containing the index) is returned into the message. This index is used on subsequent calls to CEEMGET to retrieve the remaining portion of the message. A feedback code is also returned, indicating the message has been truncated. When the entire message is returned, `msg_ptr` is zero.

The `msg_ptr` contains different results based on the length of the message:

- If a message contains fewer than 80 characters, the entire message is returned on the first call. `msg_ptr` contains 0.
- If a message contains exactly 80 characters, the entire message is returned on the first call. `msg_ptr` contains 0.
- If the message is too long, CEEMGET splits it into segments. The `msg_ptr` does not contain the cumulative index for the entire message returned so far, but contains only the index into the segment that was just returned. It is up to the user of CEEMGET to maintain the cumulative count if needed. When a message is too long, the following can occur:
  - If a message contains more than 80 characters and at least one blank is contained in the first 80 characters, the string up to and including the last blank is returned on the first call.
  - If the 80th character is nonblank (even if the 81st character is a blank), `msg_ptr` contains the index of the last blank (something less than 80), and the next call starts with the next character.
  - If the 80th character is a blank, `msg_ptr` contains 80 and the next call starts with the 81st character, blank or nonblank.
  - If a message contains more than 80 characters and at least the first 80 are all nonblank, the first 80 are returned and the next call does not add any blanks and starts with the 81st character. `msg_ptr` contains 80.

**fc** *(output)*
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE036</td>
<td>3</td>
<td>0102</td>
<td>An unrecognized condition token was passed to routine and could not be used.</td>
</tr>
<tr>
<td>CEE0E2</td>
<td>3</td>
<td>0450</td>
<td>The message inserts for the condition token with message number <code>message-number</code> and facility ID <code>facility-id</code> could not be located.</td>
</tr>
<tr>
<td>CEE0E6</td>
<td>3</td>
<td>0454</td>
<td>The message number <code>message-number</code> could not be found for facility ID <code>facility-id</code>.</td>
</tr>
<tr>
<td>CEE0E7</td>
<td>1</td>
<td>0455</td>
<td>The message with message number <code>message-number</code> and facility ID <code>facility-id</code> was truncated.</td>
</tr>
<tr>
<td>CEE0EA</td>
<td>3</td>
<td>0458</td>
<td>The message repository <code>repository-name</code> could not be located.</td>
</tr>
</tbody>
</table>
Usage notes

- z/OS UNIX considerations—In multithread applications, CEEMGET affects only the calling thread. However, CEEMGET uses the NATLANG value of the enclave. Any subsequent calls to CEEMGET, for a given condition, use the NATLANG value in effect at the time of the first call.

For more information

- See “CEENCOD—Construct a condition token” on page 407 for a description of the 12-byte condition token constructed by the CEENCOD callable service.

Examples

Figure 158 on page 377 is an example of CEEMGET called by C/C++. 
/* Module/File Name: EDCMGET */

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

int main(void) {
    _VSTRING message;
    _INT4 dest, msgindx;
    _INT2 c_1, c_2, cond_case, sev, control;
    _CHAR3 facid;
    _INT4 isi;
    _CHAR80 msgarea;
    _FEEDBACK fc, token;

    /* construct a token for CEE message 2523 */
    c_1 = 1;
    c_2 = 2523;
    cond_case = 1;
    sev = 1;
    control = 1;
    memcpy(facid, "CEE", 3);
    isi = 0;
    CEENCOD(&c_1, &c_2, &cond_case, &sev, &control, 
            facid, &isi, &token, &fc);

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

    msgindx = 0;
    memset(msgarea, ' ', 79); /* initialize the message area */
    msgarea_80 = '\0';

    /* use CEEMGET until all the message has been */
    /* retrieved */
    /* msgindx will be zero when all the message has */
    /* been retrieved */
    do {
        CEEMGET(&token, msgarea, &msgindx, &fc);
        if (fc.tok_sev > 1) {
            printf("CEEMGET failed with message number %d\n", fc.tok_msgno);
            exit(2999);
        }
    }

} /* Module/File Name: EDCMGET */
CEEMGET

/* 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);

Figure 158. C/C++ Example of CEEMGET (Part 2 of 2)

CBL LIB,QUOTE
*Module/File Name: IGZTMGET
******************************************
** **
** CBLMGET - Call CEEMGET to get a **
** message. First set up a **
** condition token using **
** CEENCOD. **
** **
******************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLMGET.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 MSGBUF PIC X(80).
  01 MSGPTR PIC S9(9) BINARY.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.
  01 NEWTOK.
    02 Condition-Token-Value.
      COPY CEENGCT.
      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 1-s-Info PIC S9(9) BINARY.

Figure 159. COBOL Example of CEEMGET (Part 1 of 2)
CEEMGET

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.

Figure 159. COBOL Example of CEEMGET (Part 2 of 2)
**CEEMGET**

Figure 160 is an example of CEEMGET called by PL/I.

```pl
/* PROCESS MACRO;  
/* Module/File Name: IBMMGET */  
/****************************************************************************/  
/** */  
/** Function : CEEMGET - Get a Message */  
/** */  
/****************************************************************************/  
PLIMGET: PROC OPTIONS(MAIN);  

INCLUDE CEEIBMMAW;  
INCLUDE CEEIBMCT;  

DCL 01 CONTOK, /* Feedback token */  
  03 MsgSev REAL FIXED BINARY(15,0),  
  03 MsgNo REAL FIXED BINARY(15,0),  
  03 Flags,  
    05 Case BIT(2),  
    05 Severity BIT(3),  
    05 Control BIT(3),  
  03 FacID CHAR(3), /* Facility ID */  
  03 ISI /* Instance-Specific Information */  
    REAL FIXED BINARY(31,0);  

DCL 01 FC, /* Feedback token */  
  03 MsgSev REAL FIXED BINARY(15,0),  
  03 MsgNo REAL FIXED BINARY(15,0),  
  03 Flags,  
    05 Case BIT(2),  
    05 Severity BIT(3),  
    05 Control BIT(3),  
  03 FacID CHAR(3), /* Facility ID */  
  03 ISI /* Instance-Specific Information */  
    REAL FIXED BINARY(31,0);  

DCL MSGBUF CHAR(80);  
DCL MSGPTR REAL FIXED BINARY(31,0);  

/* Give CONTOK value of condition CEE001 */  
ADDR( CONTOK ) -> CEEIBMCT = CEE001;  
MSGPTR = 0;  

/* Call CEEMGET to retrieve msg corresponding */  
/* to condition token */  
CALL CEEMGET( CONTOK, MSGBUF, MSGPTR, FC );  
IF FBCHECK( FC, CEE000 ) THEN DO;  
  PUT SKIP LIST( 'Message text for message number'  
    CONTOK.MsgNo || ' is "' || MSGBUF || '"');  
  END;  
ELSE DO;  
  DISPLAY( 'CEEMGET failed with msg '  
    FC.MsgNo );  
  STOP;  
END;  

END PLIMGET;  
```

Figure 160. PL/I Example of CEEMGET

## CEEMOUT—Dispatch a message

CEEMOUT dispatches a user-defined message string to the message file.

### Syntax

```plaintext
CEEMOUT((message_string, destination_code, fc))
```
CEEMOUT

message_string (input)
A halfword-prefixed printable character string containing the message. DBCS characters must be enclosed within shift-out (byte X'0F') shift-in (X'0E') characters. Insert data cannot be placed in the message with CEEMOUT. The halfword-prefixed message string (input) must contain only printable characters. For length greater than zero, unpredictable results will occur if the byte following the halfword prefix is X'00'.

destination_code (input)
A 4-byte binary integer. The only accepted value for destination_code is 2. Under systems other than CICS, Language Environment writes the message to the ddname of the file specified in the MSGFILE run-time option. Under CICS, the message is written to a transient data queue named CESE.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE0E0E</td>
<td>3</td>
<td>0451</td>
<td>An invalid destination code destination-code was passed to routine routine-name.</td>
</tr>
<tr>
<td>CEE0E9</td>
<td>3</td>
<td>0457</td>
<td>The message file destination ddname could not be located.</td>
</tr>
</tbody>
</table>

Usage notes
- z/OS UNIX considerations—In multithread applications, CEEMOUT affects only the calling thread. When multiple threads write to the message file, the output is interwoven by line. To group lines of output, serialize MSGFILE access (by using a mutex, for example).
- If the message file is defined with an LRECL greater than 256, and the input to CEEMOUT is greater than 256 bytes, the output results in multiple records. The first 256 bytes of each record contains output data. The remaining bytes of each record, up to the LRECL size, might contain unpredictable data.

For more information
- See "MSGFILE" on page 52 for more information about the MSGFILE run-time option.
- See z/OS Language Environment Programming Guide for more information on CESE.

Examples
Figure 161 on page 382 is an example of CEEMOUT called by C/C++. 

Chapter 4. Language Environment callable services 381
CEEMOUT

/*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);
  }
  /* */
}

Figure 161. C/C++ Example of CEEMOUT

Figure 162 on page 383 is an example of CEEMOUT called by COBOL.
Figure 162. COBOL Example of CEEMOUT

Figure 163 on page 384 is an example of CEEMOUT called by PL/I.
CEEMRCE—Move resume cursor explicit

The CEEMRCE service resumes execution of a user routine at the location established by CEE3SRP. CEEMRCE is designed to be called from a user condition handler and works only in conjunction with the CEE3SRP service.

Syntax

```plaintext
CEEMRCE(resume_token, fc)
```

**resume_token (input)**

An INT4 data type that contains a token, returned from the CEE3SRP service, representing the resume point in the user routine.
**FC (output/optional)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE07V</td>
<td>2</td>
<td>0255</td>
<td>The first parameter passed to CEEMRCE was an unrecognized label.</td>
</tr>
</tbody>
</table>

**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**


**Examples**

Figure 164 on page 386 is an example of CEEMRCE called by COBOL.
IDENTIFICATION DIVISION.
PROGRAM-ID. PGM2.
  *Module/File Name: IGZTMRCE

  * Sample program using CEE3SRP and CEEMRCE.*
  * PGM2 registers user-written condition
  * handler UCH1 using CEEMRCE. It
  * sets a resume point using CEE3SRP. After
  * incurring a condition and returning
  * to PGM2, PGM3 is called. PGM3 sets up
  * new resume point, does a divide-by-zero,
  * and after resuming in PGM3, resets the
  * resume point to PGM2 and does a GOBACK.
  *
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 RECOVERY-AREA EXTERNAL.
    05 RECOVERY-POINT POINTER.
    05 ERROR-INDICATOR PIC X(01).
  01 UCH-ROUTINE PROCEDURE-POINTER.
    01 FIELDS.
      05 FIRST-TIME-SW PIC X(03) VALUE ' ON'.
        88 FIRST-TIME-88 VALUE ' ON'.
      05 ANSWER PIC S9(02) COMP-3 VALUE 0.
      05 UCH1 PIC X(08) VALUE 'UCH1 '.
      05 PGMS PIC X(08) VALUE 'PGM3 '.
      05 CEEHDLR PIC X(08) VALUE 'CEEHDLR '.
      05 CEE3SRP PIC X(08) VALUE 'CEE3SRP '.
      05 TOKEN PIC S9(09) BINARY.
    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.

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
    PERFORM COMPUTE-LOOP 3 TIMES.
    CALL PGM3 USING RECOVERY-AREA.
    SET RECOVERY-POINT TO NULL.
    GOBACK.

Figure 164. COBOL Example of CEEMRCE (Part 1 of 4)
**Sample program using CEE3SRP and CEEMRCE.**

PGM2 registered UCH1. This program sets a new resume point, does a divide-by-zero, and after resuming in PGM3, resets the resume point to PGM2 and does a GOBACK.

```cobol
IDENTIFICATION DIVISION.
PROGRAM-ID. PGM3.

* Set up a new resume point.
CALL CEE3SRP USING RECOVERY-POINT,
    CASE-1
    SERVICE LABEL
    IF CASE-1 NOT = LOW-VALUE
    GOBACK.
    IF ERROR-INDICATOR = 'E'
    MOVE SPACE TO ERROR-INDICATOR
    MOVE 1 TO ANSWER.
    * Application code may go here.
    COMPUTE ANSWER = 1 / ANSWER.
    * Put application code here.
    CBL NODYNAM APOST
```

```cobol
DATA DIVISION.
WORKING-STORAGE SECTION.
01 RECOVERY-AREA EXTERNAL.
   05 RECOVERY-POINT POINTER.
   05 ERROR-INDICATOR PIC X(01).
01 UCH-ROUTINE PROCEDURE-POINTER.
```

Figure 164. COBOL Example of CEEMRCE (Part 2 of 4)
CEEMRCE

01 FIELDS.
  05 FIRST-TIME-SW PIC X(03) VALUE 'ON'.
  08 FIRST-TIME-B8 PIC X(08) VALUE 'ON'.
  05 ANSWER PIC S9(02) COMP-3 VALUE 0.
  05 CEEHDLR PIC X(08) VALUE 'CEEHDLR'.
  05 CEE3SRP PIC X(08) VALUE 'CEE3SRP'.
  05 TOKEN PIC S9(09) BINARY.
  05 SEV PIC -9(05).
  05 MSG PIC -9(05).
  05 FC.
    10 CASE-1.
      15 SEVERITY PIC S9(04) BINARY.
      15 MSG-NO PIC S9(04) BINARY.
      10 SEV-CTL PIC X(01).
      10 FACILITY-ID PIC X(03).
      10 I-S-INFO PIC S9(09) BINARY.

PROCEDURE DIVISION.

PERFORM COMPUTE-LOOP 3 TIMES.
SET RECOVERY-POINT TO NULL.
GOBACK.

COMPUTE-LOOP.
  IF FIRST-TIME-B8.
    MOVE 'OFF' TO FIRST-TIME-SW
    *__________________________________________*
    **
    * Set new resume point. *
    *__________________________________________*
    CALL CEE3SRP USING RECOVERY-POINT, FC
    SERVICE LABEL
    IF CASE-1 NOT = LOW-VALUE
    GOBACK.
    IF ERROR-INDICATOR = 'E'
    MOVE SPACE TO ERROR-INDICATOR
    MOVE 1 TO ANSWER.
    * Application code may go here.
    COMPUTE ANSWER = 1 / ANSWER.
    * Put application code here.
    CBL NODYNAM APOST
    IDENTIFICATION DIVISION.
    PROGRAM-ID. UCH1.

Figure 164. COBOL Example of CEEMRCE (Part 3 of 4)
Figure 164. COBOL Example of CEEMRCE (Part 4 of 4)

`CEEMRCE`

---

* Sample user condition handler using CEEMRCE. This program sets an error flag for the program-in-error to query and issues a call to CEEMRCE to return control to the statement following the call to CEE3SRP.

```
* ____________________________________________ *
* Call CEEMRCE to return control to the last resume point. *
* ____________________________________________ *
```

```cobol
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 USING RECOVERY-POINT,
      FC.
   IF CASE-1 NOT = LOW-VALUE
      GOBACK.
   MOVE +10 TO RESULT-CODE.
   GOBACK.
```

Figure 165 on page 390 is an example of CEEMRCE called by PL/I.
CEEMRCE

* Process lc(101), opt(0), s, map, list, stmt, a(f), ag;
* Process macro;
DRV3SRP: Proc Options(Main);

/* Module/File Name: IBM3SRP */
/*****************************************************************************/
** ** DRV3SRP - Set an explicit resume point by **
** calling CEE3SRP then registering a **
** condition handler that calls CEEMRCE **
** to resume at the explicitly set **
** resume point. **
** **
/*****************************************************************************/

%include CEEIBMCT;
%include CEEIBMAW;
declare 01 FBCODE feedback; /* Feedback token */
declare DENOMINATOR fixed binary(31,0);
declare NUMERATOR fixed binary(31,0);
declare RATIO fixed binary(31,0);
declare PLI3SRP external entry;
declare U_PTR pointer;
declare 01 U_DATA,
   03 U_CNTL fixed binary(31),
   03 U_TOK pointer;
U_PTR = addr(U_DATA);
U_CNTL = 0;

/* Set Resume Point */
Display('Setting resume point via CEE3SRP');
Call CEE3SRP(U_TOK,FBCODE);
Display('After CEE3SRP ... Resume point');
If U_CNTL = 0 Then
   Do;
      Display('First time through...');
      Display('Registering user handler');
      Call CEEHDLR(PLI3SRP, U_PTR, FBCODE);
      If FBCHECK(FBCODE, CEE000) Then
         Do;
            /* Cause a zero-divide condition */
            DENOMINATOR = 0;
            NUMERATOR = 1;
            RATIO = NUMERATOR / DENOMINATOR;
         End;
      Else
         Do;
            Display('CEEHDLR failed with msg ');
            Display(MsgNo);
         End;
      End;
   Else
      Do;
         Display('Second time through...');
      End;
   End;
/* Unregister handler */
CEEMRCR—Move resume cursor

CEEMRCR moves the resume cursor to a position relative to the current position of the handle cursor. The actions supported are:

- Moving the resume cursor to the call return point of the routine registering the executing condition handler.
- Moving the resume cursor to the caller of the routine registering the executing condition handler.

Initially, the resume cursor is placed after the instruction that caused the condition. Whenever CEEMRCR moves the resume cursor and passes stack frames, associated exit routines are invoked. Note that "exit routine" refers to user condition handlers as well as language-specific condition handlers. In addition, any associated user condition handlers are unregistered. The movement direction is always toward earlier stack frames, never toward more recent stack frames. The movement occurs only after the condition handler returns to the Language Environment condition manager.

Multiple calls to CEEMRCR yield the net results of the calls; that is, if two calls move the resume cursor to different places for the same stack frame, the most restrictive call (that closest to the earliest stack frame) is used for that stack frame.
## CEEMRCR

Moving the resume cursor to a particular stack frame:
- Cancels all stack frames from the previous resume point up to but not including the new resume point
- Unregisters any user condition handlers registered for the canceled stack frames

### Syntax

```
CEEMRCR(---type_of_move---,---fc---)
```

**type_of_move (input)**
A fullword binary signed integer indicating the target of the resume cursor movement. The possible values for `type_of_move` are:

- **0** Move the resume cursor to the call return point of the stack frame associated with the handle cursor.
- **1** Move the resume cursor to the call return point of the stack frame prior to the stack frame associated with the handle cursor. The handle cursor is moved to the most recently established condition handler of the stack frame. The new resume cursor position now points to this condition handler for the stack frame. Do not use a `type_of_move` value of 1 if the caller of the stack frame associated with the handle cursor is a nested COBOL program.

Modifying the resume cursor to point to stack frame 0 is not allowed. You cannot move the resume cursor beyond the earliest stack frame.

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE07U</td>
<td>1</td>
<td>0254</td>
<td>The first parameter passed to CEEMRCR was not 0 or 1.</td>
</tr>
<tr>
<td>CEE083</td>
<td>3</td>
<td>0259</td>
<td>A move to stack frame zero using CEEMRCR was attempted from a MAIN routine.</td>
</tr>
<tr>
<td>CEE084</td>
<td>3</td>
<td>0260</td>
<td>No condition was active when a call to a condition management routine was made. The requested function was not performed.</td>
</tr>
<tr>
<td>CEE08L</td>
<td>1</td>
<td>0277</td>
<td>CEEMRCR was called to perform an unnecessary move.</td>
</tr>
</tbody>
</table>

### Usage notes
- **PL/I MTF consideration**—CEEMRCR is not supported in PL/I MTF applications. This includes any CEEHDLR service called from a COBOL program in the application.
- **z/OS UNIX considerations**—In multithread applications, CEEMRCR affects only the calling thread. You can use CEEMRCR only within the thread’s call chain.

### Illustration of CEEMRCR Usage
The following three figures illustrate how you can move the resume cursor by using the CEEMRCR service.
In Figure 166, routine A calls routine B, which in turn calls C, which calls D. User condition handlers are registered in routines B and C.

When a condition is raised in routine D, the Language Environment condition manager passes control to the user condition handler established for routine C. The handle cursor now points to the stack frame for routine C. Routine C percolates the condition.

The handle cursor now points to the stack frame for routine B. The next user condition handler to gain control is that one established for routine B; it recognizes the condition and issues a resume by calling CEEMRCR.

A 0 type of move, meaning move the resume cursor to the stack frame associated with the handle cursor, causes control to resume at the call return point in routine B, the instruction immediately following the call to routine C. A 1 type of move, meaning move the resume cursor to the call return point of the stack frame immediately preceding the one to which the handle cursor points, moves the resume cursor to the instruction immediately following a call in routine B to routine A.

The same scenario is illustrated in Figure 167 on page 394 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
In Figure 168 on page 395, the user condition handlers are established for routines C and D. When a condition is raised in routine D, only a 1 \textit{type_of_move} is permitted. A 0 \textit{type_of_move} results in error warning message CEE0277.
Examples

Figure 169 on page 396 is an example of CEEMRCR called by C/C++. 

Figure 168. Third Example Moving Resume Cursor Using CEEMRCR
void b(void);

int main(void) {
    b();
}

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;
}

Figure 169. C/C++ Example of CEEMRCR (Part 1 of 2)
CEEMRCR

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);
is1 = 0;
CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
        facid,&is1,&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;
}

Figure 169. C/C++ Example of CEEMRCR (Part 2 of 2)

Figure 170 on page 398 is an example of CEEMRCR called by COBOL.
Module/File Name: IGZTMRCR

************************************************
** CBLMAIN - Main for sample program for CEEMRCR. **
** ************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLMAIN.
PROCEDURE DIVISION.
   CALL "DRVMRCR"
   DISPLAY "Resumed execution in the CALLER "
   "of the routine which registered the "
   "handler"
   GOBACK.
END PROGRAM CBLMAIN.

************************************************
** DRVMRCR - Drive sample program CEEMRCR. **
** ************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRVMRCR.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE PROCEDURE-POINTER.
  01 TOKEN PIC S9(9) BINARY.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
        02 I-S-Info PIC S9(9) BINARY.
        01 QDATA PIC S9(9) BINARY.

Figure 170. COBOL Example of CEEMRCR (Part 1 of 4)
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.

Figure 170. COBOL Example of CEEMRCR (Part 2 of 4)
CEEMRCR

******************************************
** 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.

Figure 170. COBOL Example of CEEMRCR (Part 3 of 4)
Figure 170. COBOL Example of CEEMRCR (Part 4 of 4)
CEEMSG—Get, format, and dispatch a message

CEEMSG gets, formats, and dispatches a message corresponding to an input condition token received from a callable service or passed to a user-written condition handler. You can use this service to print a message after a call to any Language Environment service that returns a condition token.

Figure 171. EXCOND Program (PL/I) to Handle Divide-by-Zero Condition
CEEMSG

Syntax

```
CEEMSG(—cond_token—,—destination_code—,—fc—)
```

**cond_token (input)**
A 12-byte condition token received as the result of a Language Environment callable service.

**destination_code (input)**
A 4-byte binary integer. `destination_code` can be specified only as 2, meaning write the message to the `ddname` of the file specified in the `MSGFILE` run-time option.

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0E2</td>
<td>3</td>
<td>0450</td>
<td>The message inserts for the condition token with message number <code>message-number</code> and facility ID <code>facility-id</code> could not be located.</td>
</tr>
<tr>
<td>CEE0E3</td>
<td>3</td>
<td>0451</td>
<td>An invalid destination code <code>destination-code</code> was passed to routine <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE0E6</td>
<td>3</td>
<td>0454</td>
<td>The message number <code>message-number</code> could not be found for facility ID <code>facility-id</code>.</td>
</tr>
<tr>
<td>CEE0E9</td>
<td>3</td>
<td>0457</td>
<td>The message file destination <code>ddname</code> could not be located.</td>
</tr>
<tr>
<td>CEE0EA</td>
<td>3</td>
<td>0458</td>
<td>The message repository <code>repository-name</code> could not be located.</td>
</tr>
<tr>
<td>CEE3CT</td>
<td>3</td>
<td>3485</td>
<td>An internal message services error occurred while locating the message number within a message file.</td>
</tr>
<tr>
<td>CEE3CU</td>
<td>3</td>
<td>3486</td>
<td>An internal message services error occurred while formatting a message.</td>
</tr>
<tr>
<td>CEE3CV</td>
<td>3</td>
<td>3487</td>
<td>An internal message services error occurred while locating a message number within the ranges specified in the repository.</td>
</tr>
</tbody>
</table>

**Usage notes**

- **z/OS UNIX considerations**—In multithread applications, CEEMSG affects only the calling thread. When multiple threads write to the message file, the output is interwoven by line. To group lines of output, serialize MSGFILE access (by using a mutex, for example).

**For more information**

- See "CEENCOD—Construct a condition token" on page 407 for more information about the CEENCOD callable service.
- See "MSGFILE" on page 52 for more information about the MSGFILE run-time option.
CEEMSG

Examples
Figure 172 is an example of CEEMSG called by C/C++.

/*Module/File Name: EDCMSG */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
    _VSTRING message;
    _INT4 dest,msgindx;
    _CHAR80 msgarea;
    _FEEDBACK fc,token;

    strcpy(message.string,"This is a test message");
    message.length = strlen(message.string);
    dest = 5; /* invalid dest so CEEMOUT will fail */

    CEEMOUT(&message,&dest,&fc);

    if ( _FBCHECK { fc , CEE000 } != 0 ) {
        /* put the message if CEEMOUT failed */
        dest = 2;
        CEEMSG(&fc,&dest,NULL);
        exit(2999);
    }
}

Figure 172. C/C++ Example of CEEMSG

Figure 173 on page 405 is an example of CEEMSG called by COBOL.
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLMSG.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 COUNTRY PIC X(2).
  01 DECSEP PIC X(2).
  01 MSGDEST PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.
    01 FC2.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBL3MDS.

*******************************************************************************
** Call a Lang Env svc, CEE3MDS in this case. **
** to receive a condition token that CEEMSG **
** can format as a message. Specify an **
** invalid value for country code so that a **
** condition will be built **
*******************************************************************************
MOVE "LN" TO COUNTRY.
CALL "CEE3MDS" USING COUNTRY, DECSEP, FC.
PARA-CBLMSG.

Figure 173. COBOL Example of CEEMSG (Part 1 of 2)
** 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 **
** CEE3MDS as the input condition token. **
*******************************************************************************
CALL "CEEMSG" USING FC, MSGDEST, FC2.
IF NOT CEE000 of FC2 THEN
  DISPLAY "CEEMSG failed with msg "
  Msg-No of FC2 UPON CONSOLE
  STOP RUN
END-IF.
GOBACK.

Figure 173. COBOL Example of CEEMSG (Part 2 of 2)

*PROCESS LANG_LVL(SAA), MACRO;
/* Module/ File Name: IBM_MSG */
 /*******************************************************************************/
/** */
/** Function: CEEMSG - get, format and dispatch */
/** a message */
/** */
/** In this example, CEE3MDS is called with an */
/** invalid country code so that a condition token */
/** would be returned to use as input to CEEMSG. */
/** Any LE/370 could have been called. CEEMSG uses */
/** the condition to get, format and dispatch the */
/** message associated with the condition that */
/** occurred in CEE3MDS */
/** */
/*******************************************************************************/
PLIMSG: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL COUNTRY CHARACTER (2);
DCL DECSEP CHARACTER (2);
DCL FC, /* Feedback token */
  01 fc,
  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);
CEENCOD—Construct a condition token

CEENCOD dynamically constructs a 12-byte condition token that communicates a condition in Language Environment. The condition token communicates with the Language Environment message and condition handling callable services, and user routines. Also, all Language Environment callable services use the condition-token data type to return information to the user as a feedback code.

Syntax

```
CEENCOD(-c_1,-c_2,-case,-severity,-control,-facility_ID,-i_s_info,-cond_token,-fc-)
```

**c_1 (input)**

*c_1* and *c_2* together make up the condition_ID portion of the condition token. *c_1* is a 2-byte binary integer representing the value of the first 2 bytes of the 4-byte condition_ID. For case 1, *c_1* represents the severity; for case 2, it is the class_code.

**c_2 (input)**

A 2-byte binary integer representing the value of the second 2 bytes of the condition_ID. For case 1, this is the Msg_No; for case 2, it is the cause_code.
case (input)
   A 2-byte binary integer defining the format of the condition_ID portion of the
token.

severity (input)
   A 2-byte binary integer indicating the condition’s severity. For case 1
conditions, the value of this field is the same as the severity value specified in
the condition_ID. For case 1 and 2 conditions, this field is also used to test the
condition’s severity. severity can be specified with the following values:

0    Information only (or, if the entire token is 0, no information).
1    Warning—service completed, probably correctly.
2    Error detected—correction attempted; service completed, perhaps
     incorrectly.
3    Severe error—service not completed.
4    Critical error—service not completed; condition signaled. A critical
     error is a condition jeopardizing the environment. If a critical error
     occurs during a Language Environment callable service, it is always
     signaled to the condition manager instead of returning synchronously
to the caller.

ccontrol (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 CEEBLDCTX 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
ccontrol (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 Language Environment 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 Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0CH</td>
<td>3</td>
<td>0401</td>
<td>An invalid case code case-code was passed to routine routine-name.</td>
</tr>
<tr>
<td>CEE0CI</td>
<td>3</td>
<td>0402</td>
<td>An invalid control code control-code was passed to routine routine-name.</td>
</tr>
<tr>
<td>CEE0CJ</td>
<td>3</td>
<td>0403</td>
<td>An invalid severity code severity-code was passed to routine routine-name.</td>
</tr>
<tr>
<td>CEE0CK</td>
<td>1</td>
<td>0404</td>
<td>Facility ID facility-id with non-alphanumeric characters was passed to routine routine-name.</td>
</tr>
<tr>
<td>CEE0E4</td>
<td>3</td>
<td>0452</td>
<td>An invalid facility ID facility-id was passed to routine routine-name.</td>
</tr>
</tbody>
</table>

**Usage notes**

- **C/C++ considerations**—The structure of the condition token (type_FEEDBACK) is described in the leawi.h header file shipped with Language Environment. You can assign values directly to the fields of the token in the header file without using the CEENCOD service. Figure 175 shows the layout of the type_FEEDBACK condition token in the header file.

```c
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;
```

**z/OS UNIX consideration**—In multithread applications, CEENCOD affects only the calling thread.

**For more information**

- For more information about case 1 and case 2, see CEENCOD "Usage notes."

**Examples**

Figure 176 on page 410 is an example of CEENCOD called by C/C++. 
CEENCOD

#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);
    }
    /* .
    . */
}

Figure 176. C/C++ Example of CEENCOD

Figure 177 on page 411 is an example of CEENCOD called by COBOL.
CBL LIB,QUOTE

*Module/File Name: IGZTNCOD
*************************************************
**
** CBLNCOD - Call CEENCOD to construct a
** condition token
**
**
*************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLNCOD.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.
  01 NEWTOK.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
          03 Case-2-Condition-ID
            REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
          03 Case-Sev-Ctl PIC X.
          03 Facility-ID PIC XXX.
          02 I-S-Info PIC S9(9) BINARY.
    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.

Figure 177. COBOL Example of CEENCOD (Part 1 of 2)
** 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.

Figure 177. COBOL Example of CEENCOD (Part 2 of 2)

Figure 178 on page 413 is an example of CEENCOD called by PL/I.
CEENCOD

```pli
*PROCESS MACRO;
/* Module/File Name: IBMNCOD */
/*******************************
/**
/** Function: CEENCOD - construct a condition token */
/**
*******************************/
PLINCOD: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMMAW;
%INCLUDE CEEIBMCT;

DCL SEV REAL FIXED BINARY(15,0);
DCL MSGNO REAL FIXED BINARY(15,0);
DCL CASE REAL FIXED BINARY(15,0);
DCL SEV2 REAL FIXED BINARY(15,0);
DCL CNTRL REAL FIXED BINARY(15,0);
DCL FACID CHARACTER(3);
DCL ISINFO REAL FIXED BINARY(31,0);

DCL 01 NEWTOK, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);

DCL 01 FC, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);

SEV = 0; /* Set severity portion of */
/* Condition_ID to 0, or */
/* information only. */
MSGNO = 1; /* Set msg number portion of */
/* Condition_ID to 1. */
CASE = 1; /* Set case to 1. This is a */
/* service condition. */
SEV2 = 0; /* Set severity to 0, or */
/* information only. */
CNTRL = 0; /* Set control to 0, or Facility */
/* ID has been assigned by user */
FACID = 'USR'; /* Set Facility_ID to USR for a */
/* user condition token. */
ISINFO = 0; /* Set I_S_Info to 0, indicating */
/* that no Instance Specific */
/* information is to be supplied. */
```

Figure 178. PL/I Example of CEENCOD (Part 1 of 2)
CEEQCEN

CEEQCEN—Query the century window

CEEQCEN queries the century in which Language Environment contains the
2-digit year value. When you want to change the setting, use CEEQCEN to get the
setting and then use CEESCEN to save and restore the current setting.

Syntax

```plaintext
CEEQCEN(—century_start—,—fc—)
```

```
 century_start (output)
 An integer between 0 and 100 indicating the year on which the century
 window is based. For example, if the Language Environment default is in
 effect, all 2-digit years lie within the 100-year window starting 80 years prior
to the system date. CEEQCEN then returns the value 80. An 80 value indicates
to Language Environment that, in 1995, all 2-digit years lie within the 100-year
window starting 80 years before the system date (between 1915 and 2014,
inclusive).
```

```
 fc (output)
 A 12-byte feedback code, optional in some languages, that indicates the result
 of this service. The following Feedback Code can result from this service:
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

Usage notes

- z/OS UNIX considerations—CEEQCEN applies to the enclave, as does the
century window.

For more information

- See [CEESCEN—Set the century window](page 428) for more information
about the CEESCEN callable service.

Examples

Figure 179 on page 415 is an example of CEEQCEN called by C/C++.
Figure 179. C/C++ Example of CEEQCEN

Figure 180 on page 416 is an example of CEEQCEN called by COBOL.
Figure 180. COBOL Example of CEEQCEN

Figure 181 on page 417 is an example of CEEQCEN called by PL/I.
CEEQDTC

CEEQDTC, analogous to the C language function `localdtconv()`, queries the date and time conventions from the current locale and sets the components of a structure with values appropriate to the settings for the LC_TIME category. CEEQDTC is sensitive to the locales set by `setlocale()` or `CEESETL`, not to the Language Environment settings from COUNTRY or CEE3CTY.

**Syntax**

```plaintext
>>CEEQDTC((--version--,--localdt--,--fc--)...) <<<
```

---

Figure 181. PL/I Example of CEEQCEN

**CEEQDTC—Query locale date and time conventions**

CEEQDTC, analogous to the C language function `localdtconv()`, queries the date and time conventions from the current locale and sets the components of a structure with values appropriate to the settings for the LC_TIME category. CEEQDTC is sensitive to the locales set by `setlocale()` or `CEESETL`, not to the Language Environment settings from COUNTRY or CEE3CTY.
CEEQDTC

version
This parameter points to a user supplied VERSION_INFO structure, in which the first four bytes contain the callable service version number.

Requirement: If the parameter does not point to a user supplied VERSION_INFO structure, it must be omitted. If the parameter points to a VERSION_INFO containing version number 1, CEEQDTC returns a DTCONV structure with the new element showing the format of the ISO 8601:2000 standard date. If the parameter is omitted, CEEQDTC returns the DTCONV format used before C99. For information on how to code this parameter, see "Invoking callable services" on page 112.

localdt (output)
A pointer to the data structure containing the date and time formatting information from the current, active locale. The fields used to populate the structure come from the LC_TIME category. The LC_TIME category structure fields used to retrieve the date and time values are:

abmon  Abbreviated month names (12 instances of a halfword length-prefixed character string, VSTRING, of 22 bytes)
mon    Month names (12 instances of a halfword length-prefixed character string, VSTRING, of 62 bytes)
abday  Abbreviated day names (7 instances of a halfword length-prefixed character string, VSTRING, of 22 bytes)
day    Day names (7 instances of a halfword length-prefixed character string, VSTRING, of 62 bytes)
d_t_fmt  Date and time format (1 instance of a halfword length-prefixed character string, VSTRING, of 82 bytes)
d_fmt  Date format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)
t_fmt  Time format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)
am_fmt  AM string (1 instance of a halfword length-prefixed character string, VSTRING, of 22 bytes)
pm_fmt  PM string (1 instance of a halfword length-prefixed character string, VSTRING, of 22 bytes)
t_fmt_ammp  Time format ampm (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)
ISO_STD8601_2000  ISO8601:2000 standard date format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)

fc (output/optional)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
<tr>
<td>CEE3VN</td>
<td>3</td>
<td>4087</td>
<td>Input Error: The version_info control block contains a version number that is not valid.</td>
</tr>
</tbody>
</table>
Usage notes

- PL/I MTF consideration—CEEQDTC is not supported in PL/I MTF applications.
- If no call to CEESETL has been made, the default locale values to be used are determined at installation time for Language Environment.
- This callable service uses the C/C++ run-time library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- See “CEESETL—Set locale operating environment” on page 448 for more information about the current locale.

Examples

Figure 182 on page 420 is an example of CEEQDTC called by COBOL.
**IDENTIFICATION DIVISION.**
**PROGRAM-ID. MAINQDTC.**
**DATA DIVISION.**
**WORKING-STORAGE SECTION.**
* Use DTCONV structure for CEEQDTC calls
COPY CEEIGZDT.
* **PROCEDURE DIVISION.**
* Subroutine needed for addressing
CALL "COBQDTC" USING DTCONV.
STOP RUN.
* **IDENTIFICATION DIVISION.**
**PROGRAM-ID. COBQDTC.**
**DATA DIVISION.**
**WORKING-STORAGE SECTION.**
01 Locale-Name.
  02 LN-Length PIC S9(4) BINARY.
  02 LN-String PIC X(256).
* Use Locale category constants
COPY CEEIGZLC.
* 01 Test-Length1 PIC S9(4) BINARY.
  01 Test-String1 PIC X(80).
  01 Test-Length2 PIC S9(4) BINARY.
  01 Test-String2 PIC X(80).
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
    03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
* **LINKAGE SECTION.**
* Use Locale structure DTCONV for CEEQDTC calls
COPY CEEIGZDT.
* **PROCEDURE DIVISION USING DTCONV.**
* Set up locale for France
MOVE 4 TO LN-Length.
MOVE "FFEY" TO LN-String (1:LN-Length).

Figure 182. COBOL Example of CEEQDTC (Part 1 of 2)
Figure 183 on page 422 is an example of CEEQDTC called by PL/I.

Figure 182. COBOL Example of CEEQDTC (Part 2 of 2)
**CEEQDTC**

*PROCESS MACRO;
/*Module/File Name: IBMQDTC */
******************************************************************************
/* Example for callable service CEEQDTC */
/* Function: Retrieve date and time convention */
/* structures for two countries, compare an item. */
******************************************************************************

PLIQDTC: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs */
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants */
%INCLUDE CEEIBMLC; /* Locale category constants */
%INCLUDE CEEIBMDT; /* DTCONV for CEEQDTC calls */

/* use explicit pointer to local DTCONV structure */
DCL LOCALDT POINTER INIT(ADDR(DTCONV));

/* CEESETL service call arguments */
DCL LOCALE_NAME CHAR(256) VARYING;
DCL 1 DTCONVC LIKE DTCONV; /* Def Second Structure */
DCL 1 FC, /* Feedback token */
  3 MsgSev REAL FIXED BINARY(15,0),
  3 MsgNo REAL FIXED BINARY(15,0),
  3 Flags,
  5 Case BIT(2),
  5 Severity BIT(3),
  5 Control BIT(3),
  3 FacID CHAR(3), /* Facility ID */
  3 ISI /* Instance-Specific Information */
REAL FIXED BINARY(31,0);

/* set locale with IBM default for France */
LOCALE_NAME = 'FFEY'; /* or Fr_FR.IBM-1047 */

/* use LC_ALL category constant from CEEIBMLC */
CALL CEESETL ( LOCALE_NAME, LC_ALL, FC );

/* retrieve date and time structure, France Locale */
CALL CEEQDTC ( *, LOCALDT, FC );

Figure 183. PL/I Example of CEEQDTC (Part 1 of 2)
CEEQRYL

/* 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;
END PLIQDTC;

Figure 183. PL/I Example of CEEQDTC (Part 2 of 2)

CEEQRYL—Query active locale environment

CEEQRYL, analogous to the C language function localename=setlocale(category, NULL), queries the environment for which locale defines the current setting for the locale category. CEEQRYL is sensitive to the locales set by setlocale() or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

category (input)
A symbolic integer number that represents all or part of the locale for a program. Depending on the value of the localename, these categories can be initiated by the values of global categories of corresponding names. The following values for the category parameter are valid; Language Environment locale callable services do not support the LC_TOD and LC_SYNTAX categories.

**LC_ALL**
A 4-byte integer (with a value of -1) that affects all locale categories associated with a program’s locale.

**LC_COLLATE**
A 4-byte integer (with a value of 0) that affects the behavior of regular expression and collation subroutines.

**LC_CTYPE**
A 4-byte integer (with a value of 1) that affects the behavior of regular expression, character classification, case conversion, and wide character subroutines.

**LC_MESSAGES**
A 4-byte integer (with a value of 6) that affects the format and values for positive and negative responses.
**CEEQRYL**

**LC_MONETARY**
A 4-byte integer (with a value of 3) that affects the behavior of subroutines that format monetary values.

**LC_NUMERIC**
A 4-byte integer (with a value of 2) that affects the behavior of subroutines that format non-monetary numeric values.

**LC_TIME**
A 4-byte integer (with a value of 4) that affects the behavior of time conversion subroutines.

**localename (output)**
Returns the name of the locale that describes the current setting of the category requested.

**fc (output/optional)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2KD</td>
<td>3</td>
<td>2701</td>
<td>An invalid category parameter was passed to a locale function.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
</tbody>
</table>

**Usage notes**

- **PL/I MTF consideration**—CEEQRYL is not supported in PL/I MTF applications.
- **CEEQRYL does not change the status of the active locales.**
- **This callable service uses the C/C++ run-time library.** The C/C++ library must be installed and accessible even when this service is called from a non-C program.
- **If the active locale is not explicitly set with CEESETL or setlocale(category, localename), then the locale chosen is as follows:**
  - With POSIX(OFF), the SAA C locale is chosen, and querying the locale with CEEQRYL returns “C” as the locale name.
  - With POSIX(ON), the POSIX C locale is chosen, and querying the locale with CEEQRYL returns “POSIX” as the locale name.

The SAA C locale provides compatibility with previous releases of C/370. The POSIX C locale provides consistency with POSIX requirements and supports the z/OS UNIX System Services environment.

**For more information**
- See “CEESETL—Set locale operating environment” on page 448 for more information about the current locale.
- For more information about LC_TIME, see “CEEQDTC—Query locale date and time conventions” on page 417.
- For information on the definition of the SAA C and POSIX C locales and the differences between them, see z/OS XL C/C++ Programming Guide

**Examples**
For examples of how to use CEEQRYL with other Language Environment locale callable services, see “CEESETL—Set locale operating environment” on page 448.
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:

\[
\text{seed}(i) = (950706376 \times \text{seed}(i-1)) \mod 2147483647;
\]

\[
\text{randomno}(i) = \text{seed}(i) / 2147483647;
\]

Syntax

```plaintext
CEERAN0(—seed—,—random_no—,—fc—)
```

**seed** *(input/output)*

A fullword binary signed integer representing an initial value used to generate random numbers. *seed* must be a variable; it cannot be an input-only parameter. The valid range is 0 to +2,147,483,646. If *seed* equals 0, the seed is generated from the current Greenwich Mean Time. On return to the calling routine, CEERAN0 changes the value of *seed* so that it can be used as the new seed in the next call.

**random_no** *(output)*

An 8-byte double precision floating-point number with a value between 0 and 1, exclusive. If *seed* is invalid, *random_no* is set to -1 and CEERAN0 terminates with a non-CEE000 symbolic feedback code.

**fc** *(output)*

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2ER</td>
<td>1</td>
<td>2523</td>
<td>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.</td>
</tr>
<tr>
<td>CEE2ES</td>
<td>3</td>
<td>2524</td>
<td>An invalid seed value was passed to CEERAN0. The random number was set to -1.</td>
</tr>
</tbody>
</table>

**Usage notes**

- *z/OS UNIX considerations*—In multithread applications, CEERAN0 affects only the calling thread. The seed is unique to the thread.

**Examples**

Figure 184 on page 426 is an example of CEERAN0 called by C/C++. 
CEERAN0

```c
/*Module/File Name: EDCRAN0 */

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

int main (void) {
    _INT4 seed;
    _FLOAT8 random;
    _FEEDBACK fc;
    int number;

    seed = 0;
    CEERAN0(&seed,&random,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEERAN0 failed with message number \%d\n",fc.tok_msgno);
        exit(2999);
    }
    number = random * 1000;
    printf("The 3 digit random number is \%d\n",number);
}
```

Figure 184. C/C++ Example of CEERAN0

Figure 185 on page 427 is an example of CEERAN0 called by COBOL.
CEERAN0

Figure 185. COBOL Example of CEERAN0

Figure 186 on page 428 is an example of CEERAN0 called by PL/I.
CEESCEN—Set the century window

CEESCEN sets the century in which Language Environment contains the 2-digit year value. Use it in conjunction with CEEDAYS or CESECS when:

- You process date values containing 2-digit years (for example, in the YYMMDD format).
- The Language Environment default century interval does not meet the requirements of a particular application.

Century intervals are kept as thread-level data, so changing the interval in one thread does not affect the interval in another thread. To query the century window, use CEEQCEN.

CEEQCEN affects and is affected by only the Language Environment NLS and date and time services, not the Language Environment locale callable services or the C locale-sensitive functions.

*PROCESS MACRO;
/* Module/File Name: IBMRAN0 */
************************************************************************/
/**/
/** Function: CEERAN0 - calculate uniform random */
/** numbers */
/** */
************************************************************************/
PLIRAN0: PROC OPTIONS(MAIN);

%INCLUDE CEEIBM4W;
%INCLUDE CEEIBMCT;

DCL SEED REAL FIXED BINARY(31,0);
DCL RANDNUM REAL FLOAT DECIMAL(16);
DCL 01 FC, /* Feedback token */
03 MsgSev REAL FIXED BINARY(15,0),
03 MsgNo REAL FIXED BINARY(15,0),
03 Flags,
05 Case BIT(2),
05 Severity BIT(3),
05 Control BIT(3),
**
03 FacID CHAR(3), /* Facility ID */
03 ISI /* Instance-Specific Information */
REAL FIXED BINARY(31,0);

SEED = 7; /* Specify an integer as the initial */
/* value used to calculate the random */
/* numbers */

/* Call CEERAN0 to generate random number between */
/* 0.0 and 1.0 */
CALL CEERAN0 ( SEED, RANDNUM, FC );

IF FBCHECK( FC, CEE000) THEN DO;
   PUT SKIP LIST ( 'The random number is ' || RANDNUM );
END;
ELSE DO;
   DISPLAY( 'CEERAN0 failed with msg ' || FC.MsgNo );
   STOP;
END;

END PLIRAN0;

Figure 186. PL/I Example of CEERAN0
Syntax

\[
\text{CEERAN0}(\text{--century\_start--}, \text{--fc--})
\]

century_start

An integer between 0 and 100, setting the century window. A value of 80, for example, places all two-digit years within the 100-year window starting 80 years before the system date. In 1995, therefore, all two-digit years are assumed to represent dates between 1915 and 2014, inclusive.

fc (output)

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2E6</td>
<td>3</td>
<td>2502</td>
<td>The UTC/GMT was not available from the system.</td>
</tr>
</tbody>
</table>

Usage notes

- z/OS UNIX considerations—CEESCEN applies to the enclave. The century window applies to the enclave.

For more information

- See “CEEDAYS—Convert date to Lilian format” on page 250 for more information about the CEEDAYS callable service.
- See “CEESECS—Convert timestamp to seconds” on page 441 for more information about the CEESECS callable service.
- See “CEEQCEN—Query the century window” on page 414 for more information about the CEEQCEN callable service.

Examples

Figure 187 on page 430 is an example of CEESCEN called by C/C++. 
CEESCEN

/*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);
}
}

Figure 187. C/C++ Example of CEESCEN

Figure 188 on page 431 is an example of CEESCEN called by COBOL.
**CBLSCEN - Call CEESCEN to set the Lang. Env. century window**

**In this example, CEESCEN is called to change**
**the start of the century window to 30 years**
**before the system date. CEESCEN 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.
   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-CBLSCEN.

Figure 188. COBOL Example of CEESCEN (Part 1 of 2)
Figure 189 is an example of CEESCEN called by PL/I.

CEESCEN—Compare collation weight of two strings

CEESCEN, which is analogous to the C language function `strcoll()`, compares two character strings based on the collating sequence specified in the LC_COLLATE category of the current locale. CEESCEN associates a collation
weight with every character in the code set for the locale. The characters in the input strings are converted to their collation weights and then compared on the basis of these weights. CEESCOL is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

### Syntax

```c
CEESCOL(omitted_parm,string1,string2,result,fc)
```

**omitted_parm**

This parameter is reserved for future expansion and must be omitted. For information on how to code an omitted parameter, see “Invoking callable services” on page 112.

**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/optional)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
</tbody>
</table>

### Usage notes

- **PL/I MTF consideration**—CEESCOL is not supported in PL/I MTF applications.
- This callable service uses the C/C++ run-time library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

### For more information

- See “CEESETL—Set locale operating environment” on page 448 for more information about the CEESETL callable service.
Examples

Figure 190 is an example of CEESCOL called by COBOL.

CBL LIB,QUOTE
*Module/File Name: IGZTSCOL
*************************************************
* Example for callable service CEESCOL *
* COBSCOL - Compare two character strings *
* and print the result. *
* Valid only for COBOL for MVS & VM Release 2 *
* or later. *
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSCOL.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 String1.
   02 Str1-Length PIC S9(4) BINARY.
   02 Str1-String.
      03 Str1-Char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Str1-Length.
01 String2.
   02 Str2-Length PIC S9(4) BINARY.
   02 Str2-String.
      03 Str2-Char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Str2-Length.
01 Result PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
COPY CEE1GZCT.
      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-Seq-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 "12345$789" TO Str2-String (1:Str2-Length)

Figure 190. COBOL Example of CEESCOL (Part 1 of 2)
CALL "CEESCOL" USING OMITTED, String1, String2, Result, FC.

IF Severity > 0
   DISPLAY "Call to CEESCOL failed. " Msg-No
   STOP RUN
END-IF.

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.
**CEESECI**

CEESECI—Convert seconds to integers

CEESECI converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. Use CEESECI instead of CEEDATM when the output is needed in numeric format rather than in character format.

The inverse of CEESECI is CEEISEC, which converts integer year, month, day, hour, second, and millisecond to number of seconds. If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86,400 (number of seconds in a day), and pass the new value to CEESECI.

Figure 191. PL/I Example of CEESCOL

```pli
*PROCESS MACRO;
/!*Module/File Name: IBMSCOL */
/!* Example for callable service CEESCOL */
/!* Function: Compare two character strings and */
/!* print the result. */
/!******************************************************************/

PLISCOL: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs for Language Environment */
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants */
%INCLUDE CEEIBMLC; /* Locale category constants */

/!* CEESCOL service call arguments */
DCL STRING1 CHAR(256) VARYING;/* first string */
DCL STRING2 CHAR(256) VARYING;/* second string */
DCL RESULT_SCOL BIN FIXED(31);/* result of compare */

DCL 01 FC, /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
    03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);

STRING1 = '12345a789';
STRING2 = '12346$789';

CALL CEESCOL( *, STRING1, STRING2, RESULT_SCOL,FC);

/* FBCHECK macro 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;
```
input_seconds
A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range for input_seconds is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999). If input_seconds is invalid, all output parameters except the feedback code are set to 0.

output_year (output)
A 32-bit binary integer representing the year. The range of valid output_years is 1582 to 9999, inclusive.

output_month (output)
A 32-bit binary integer representing the month. The range of valid output_months is 1 to 12.

output_day (output)
A 32-bit binary integer representing the day. The range of valid output_days is 1 to 31.

output_hours (output)
A 32-bit binary integer representing the hour. The range of valid output_hours is 0 to 23.

output_minutes (output)
A 32-bit binary integer representing the minutes. The range of valid output_minutes is 0 to 59.

output_seconds (output)
A 32-bit binary integer representing the seconds. The range of valid output_seconds is 0 to 59.

output_milliseconds (output)
A 32-bit binary integer representing milliseconds. The range of valid output_milliseconds is 0 to 999.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2E9</td>
<td>3</td>
<td>2505</td>
<td>The input_seconds value in a call to CEEDATM or CEESECI was not within the supported range.</td>
</tr>
</tbody>
</table>
Usage notes

- z/OS UNIX consideration—In multithread applications, CEESECI affects only the calling thread.

For more information

- For more information about the CEEISEC callable service, see "CEEISEC—Convert integers to seconds" on page 351.

Examples

Figure 192 is an example of CEESECI called by C/C++.

```c
/*Module/File Name: EDCSECI */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
    _INT4 year, month, day, hours, minutes, seconds,
    milliseconds;
    _FLOAT8 input;
    _FEEDBACK fc;

    input = 13166064000.0;
    CEESECI(&input,&year,&month,&day,&hours,&minutes,;
    &seconds,&milliseconds,&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
",input,hours,minutes,
    seconds,milliseconds,month,day,year);
}
```

Figure 192. C/C++ Example of CEESECI

Figure 193 on page 439 is an example of CEESECI called by COBOL.
** 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.

Figure 193. COBOL Example of CEESECI (Part 1 of 2)
**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.

**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.

---

**Figure 193. COBOL Example of CEESECI (Part 2 of 2)**

[Figure 194 on page 441] is an example of CEESECI called by PL/I.
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 CEE3CTY callable service, not the CEESETL callable service or the setlocale() function.
The inverse of CEESECS is CEEDATM, which converts `output_seconds` to character format. By default, 2-digit years lie within the 100 year range starting 80 years prior to the system date. Thus, in 1995, all 2-digit years represent dates between 1915 and 2014, inclusive. You can change this range by using the callable service CEESECN.

### Syntax

```
CEESECS(input_timestamp, picture_string, output_seconds, fc)
```

**input_timestamp (input)**

A length-prefixed character string representing a date or timestamp in a format matching that specified by `picture_string`. The character string must contain between 5 and 80 picture characters, inclusive. `input_timestamp` can contain leading or trailing blanks. Parsing begins with the first nonblank character (unless the picture string itself contains leading blanks; in this case, CEESECS skips exactly that many positions before parsing begins).

After a valid date is parsed, as determined by the format of the date you specify in `picture_string`, all remaining characters are ignored by CEESECS. Valid dates range between and including the dates 15 October 1582 to 31 December 9999. A full date must be specified. Valid times range from 00:00:00.000 to 23:59:59.999.

As the following example shows, if any part or all of the time value is omitted, zeros are substituted for the remaining values.

<table>
<thead>
<tr>
<th>Date and Time</th>
<th>Equivalent Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1992-05-17-19:02</td>
<td>1992-05-17-19:02:00</td>
</tr>
<tr>
<td>1992-05-17</td>
<td>1992-05-17-00:00:00</td>
</tr>
</tbody>
</table>

**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.MMSS 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`:

```c
CALL CEESECS('92/06/03 15.35.03', 'YY/MM/DD HH.MMSS', secs, fc);
CALL CEESECS('92/6/3 15.35.03', 'YY/MM/DD HH.MMSS', secs, fc);
CALL CEESECS('92/6/3 3.35.03 PM', 'YY/MM/DD HH.MMSS AP', secs, fc);
CALL CEESECS('92.155 3.35.03 pm', 'YY.DDD HH.MMSS 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 522 for a list of Japanese eras supported by CEESECS.

If picture_string includes era symbol <CCCC> or <CCCCCCCC>, the YY position in input_timestamp represents the year number within the era. See Table 33 on page 521 for a list of valid picture characters, and Table 34 on page 522 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 Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2EB</td>
<td>3</td>
<td>2507</td>
<td>Insufficient data was passed to CEEDAYS or CEESECS. The Lilian value was not calculated.</td>
</tr>
<tr>
<td>CEE2EC</td>
<td>3</td>
<td>2508</td>
<td>The date value passed to CEEDAYS or CEESECS was invalid.</td>
</tr>
<tr>
<td>CEE2ED</td>
<td>3</td>
<td>2509</td>
<td>The era passed to CEEDAYS or CEESECS was not recognized.</td>
</tr>
<tr>
<td>CEE2EE</td>
<td>3</td>
<td>2510</td>
<td>The hours value in a call to CEEISEC or CEESECS was not recognized.</td>
</tr>
<tr>
<td>CEE2EH</td>
<td>3</td>
<td>2513</td>
<td>The input date passed in a CEEISEC, CEEDAYS, or CEESECS call was not within the supported range.</td>
</tr>
<tr>
<td>CEE2EK</td>
<td>3</td>
<td>2516</td>
<td>The minutes value in a CEEISEC call was not recognized.</td>
</tr>
<tr>
<td>CEE2EL</td>
<td>3</td>
<td>2517</td>
<td>The month value in a CEEISEC call was not recognized.</td>
</tr>
<tr>
<td>CEE2EM</td>
<td>3</td>
<td>2518</td>
<td>An invalid picture string was specified in a call to a date/time service.</td>
</tr>
<tr>
<td>Code</td>
<td>Severity</td>
<td>Message Number</td>
<td>Message Text</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CEE2EN</td>
<td>3</td>
<td>2519</td>
<td>The seconds value in a CEEISEC call was not recognized.</td>
</tr>
<tr>
<td>CEE2EP</td>
<td>3</td>
<td>2521</td>
<td>The year-within-era value passed to CEESECS was zero.</td>
</tr>
<tr>
<td>CEE2ET</td>
<td>3</td>
<td>2525</td>
<td>CEESECS detected non-numeric data in a numeric field, or the timestamp string did not match the picture string.</td>
</tr>
</tbody>
</table>

**Usage notes**

- The probable cause for receiving message number 2518 is a picture string that contains an invalid DBCS string. You should verify that the data in the picture string is correct.
- z/OS UNIX consideration—In multithread applications, CEESECS affects only the calling thread.

**For more information**

- See “CEESCEN—Set the century window” on page 428 for more information about the CEESCEN callable service.
- See “COUNTRY” on page 22 for more information about the COUNTRY run-time option.

**Examples**

Figure 195 on page 445 is an example of CEESECS called by C/C++.
/*Module/File Name: EDCSECS*/
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc;
    _FLOAT8 seconds1, seconds2;
    _VSTRING date,date_pic;

    /* use CEESECS to convert to seconds timestamp */
    strcpy(date.string, "09/13/91 23:23:23");
    date.length = strlen(date.string);
    strcpy(date_pic.string, "MM/DD/YY HH:MI:SS");
    date_pic.length = strlen(date_pic.string);

    CEESECS(&date, &date_pic, &seconds1, &fc);
    if ( _FBCHECK ( fc, CEE000 ) != 0 ) { 
        printf("CEESECS failed with message number %d\n",
               fc.tok_msgno);
        exit(2999);
    }

    strcpy(date.string, "December 15, 1992 at 8:23:45 AM");
    date.length = strlen(date.string);
    strcpy(date_pic.string, "Mmmm 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:
            September 13, 1991 at 11:23:23 PM
            and December 15, 1992 at 8:23:45 AM is:
            %f\n",
            seconds2 - seconds1);
}

Figure 195. C/C++ Example of CEESECS

Figure 196 on page 446 is an example of CEESECS called by COBOL.
** CEESECS - 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.

Figure 196. COBOL Example of CEESECS (Part 1 of 2)
** 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.

Figure 196. COBOL Example of CEESECS (Part 2 of 2)

Figure 197 on page 448 is an example of CEESECS called by PL/I.
CEESETL—Set locale operating environment

CEESETL, analogous to the C language function setlocale(), establishes a global locale operating environment, which determines the behavior of character collation, character classification, date and time formatting, numeric punctuation, and positive/negative response patterns. CEESETL is sensitive to the locales set by setlocale() or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

**Syntax**

```
CEESETL(—localename—,—category—,—fc—)
```
localename (input)
A halfword length-prefixed character string (VSTRING) with a maximum of 256 bytes. localename is a valid locale name known to the locale model. The category named in the call is set according to the named locale. If localename is null or blank, CEESETL sets the locale environment according to the environment variables. This is the equivalent to specifying setlocale(LC_ALL,""). If these environment variables are defined, you can locate them in the following order:
- LC_ALL if it is defined and not null
- LANG if it is defined and not null
- The default C locale

category (input)
A symbolic integer number that represents all or part of the locale for a program. Depending on the value of the localename, these categories can be initiated by the values of global categories of corresponding names. The following values for the category parameter are valid:

LC_ALL
A 4-byte integer (with a value of -1) that affects all locale categories associated with a program's locale.

LC_COLLATE
A 4-byte integer (with a value of 0) that affects the behavior of regular expression and collation subroutines.

LC_CTYPE
A 4-byte integer (with a value of 1) that affects the behavior of regular expression, character classification, case conversion, and wide character subroutines.

LC_MESSAGES
A 4-byte integer (with a value of 6) that affects the format and values for positive and negative responses.

LC_MONETARY
A 4-byte integer (with a value of 3) that affects the behavior of subroutines that format monetary values.

LC_NUMERIC
A 4-byte integer (with a value of 2) that affects the behavior of subroutines that format non-monetary numeric values.

LC_TIME
A 4-byte integer (with a value of 4) that affects the behavior of time conversion subroutines.

Language Environment locale callable services do not support the LC_TOD and LC_SYNTAX categories.

fc (output/optional)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE2KD</td>
<td>3</td>
<td>2701</td>
<td>An invalid category parameter was passed to a locale function.</td>
</tr>
</tbody>
</table>
CEESETL

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE2KE</td>
<td>3</td>
<td>2702</td>
<td>An invalid locale name parameter was passed to a locale function.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
</tbody>
</table>

**Usage notes**

- PL/I MTF consideration—CEESETL is not supported in PL/I MTF applications.
- This callable service uses the C/C++ run-time library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.
- The LC_ALL category indicates that all categories are to be changed with regard to the specific locale. The LC_ALL value, when set by CEESETL, becomes the current values for all six LC_* categories.
- If the active locale is not explicitly set with CEESETL or `setlocale(category, localename)`, then the locale chosen is as follows:
  - With POSIX(OFF), the SAA C locale is chosen, and querying the locale with `CEEQRL` returns “C” as the locale name.
  - With POSIX(ON), the POSIX C locale is chosen, and querying the locale with `CEEQRL` returns “POSIX” as the locale name.

The SAA C locale provides compatibility with previous releases of C/370. The POSIX C locale provides consistency with POSIX requirements and supports the z/OS UNIX environment.

**For more information**

- See [z/OS XL C/C++ Run-Time Library Reference](#) for details of how various locale categories affect C/C++ language functions.
- For more information about specifying environment variables to set the locale, see [z/OS XL C/C++ Programming Guide](#).
- For information on the definition of the SAA C and POSIX C locales and the differences between them, see [z/OS XL C/C++ Programming Guide](#).
- For more information about LC_TIME, see “CEEQDTC—Query locale date and time conventions” on page 417.

**Examples**

Figure 198 on page 451 is an example of CEESETL called by COBOL.
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.

Figure 198. COBOL Example of CEESETL (Part 1 of 2)
**Figure 198. COBOL Example of CEESETL (Part 2 of 2)**

**Figure 199** is an example of CEESETL called by COBOL.

```
*PROCESS MACRO;
/*Module/File Name: IBMSETL */
/*Example for callable service CEESETL */
/*Function: Set all global locale environment */
/*categories to country. Query one category. */
/**********************************************/
PLISETL: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs */
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants */
%INCLUDE CEEIBMLC; /* Locale category constants */

/* CEESETL service call arguments */
DCL LOCALE_NAME CHAR(14) VARYING;

/* CEEQRYL service call arguments */
DCL LOCALE_NAME_Time CHAR(256) VARYING;

DCL 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);
```
CEESGL—Signal a condition

CEESGL raises, or signals, a condition to the Language Environment condition manager. It also provides qualifying data and creates an ISI for a particular instance of the condition. The ISI contains information used by the Language Environment condition manager to identify and react to conditions.

CEESGL is typically used to generate application-specific conditions that are recognized by condition handlers registered using CEEHDLR. Conditions can also be selected to simulate a Language Environment or system condition. If you plan on using a routine that signals a new condition with a call to CEESGL, you should first call the CEECMII callable service to copy any insert information into the ISI associated with the condition.

CEESGL generates a Language Environment condition. You can map some of the Language Environment condition tokens to POSIX signals. Unique conditions signaled by CEESGL are considered to be enabled under Language Environment. Therefore, they undergo Language Environment condition handling.

CEESGL can signal a POSIX condition. If CEESGL signals a POSIX condition and the signal is blocked at the time of the generation but later unblocked and delivered, the POSIX signal processing semantics are applied. The Language Environment synchronous condition manager semantics do not apply.
Severity 0 and 1 conditions are considered safe conditions. They can be ignored if they are not handled and if no feedback token is passed when the condition is raised.

Each signaled condition (of severity 2 or above) increments the error count by one. If the error count exceeds the error count limit (as specified by the ERRCOUNT run-time option—see "ERRCOUNT" on page 31) the condition manager terminates the enclave with abend code 4091, reason code 11. T_I_U (Termination Imminent due to an Unhandled Condition) is not issued. Promoted conditions do not increment the error count. A program established using the CEEHDLR callable service or one of the HLL condition handlers, can then process the raised condition.

ERRCOUNT applies to CEESGL only if the condition generated by CEESGL is delivered synchronously. POSIX signal handling semantics are then applied to the condition.

Table 24 on page 209 contains a list of the S/370 program interrupt codes and their corresponding Language Environment condition token names and message numbers.

### Syntax

```
CEESGL(cond_rep, q_data_token, fc)
```

**cond_rep** (input)

A 12-byte condition token representing the condition to be signaled. You can either construct your own condition token or use one that Language Environment has already defined. Conditions signaled by CEESGL are not necessarily handled by Language Environment. If you call CEESGL with a `cond_rep`, Language Environment passes control to the language in which the routine is written. The condition manager then determines if it should handle the condition. If so, the HLL handles the condition. If not, control returns to Language Environment. The condition might also be ignored or blocked, or might result in enclave termination.

**q_data_token** (input/output)

An optional 32-bit data object placed in the ISI to access the qualifying data (q_data) associated with the given instance of the condition. The `q_data_token` is a list of information addresses a user condition handler uses to specifically identify and, if necessary, react to, a given condition. The information in the `q_data_token` provides a mechanism by which user-written condition handlers can provide a complete fix-up of some conditions. The `q_data_token` associated with a condition using CEESGL can be extracted later using the CEEQDQDT callable service.

**fc** (output / optional)

One of the following 12-byte feedback codes, optional in some languages, that indicates the result of this service.

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>Code</td>
<td>Severity</td>
<td>Message Number</td>
<td>Message Text</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CEE069</td>
<td>0</td>
<td>0201</td>
<td>An unhandled condition was returned in a feedback code.</td>
</tr>
<tr>
<td>CEE0CE</td>
<td>1</td>
<td>0398</td>
<td>Resume with new input.</td>
</tr>
<tr>
<td>CEE0CF</td>
<td>1</td>
<td>0399</td>
<td>Resume with new output.</td>
</tr>
<tr>
<td>CEE0EB</td>
<td>3</td>
<td>0459</td>
<td>Not enough storage was available to create a new instance specific information block.</td>
</tr>
<tr>
<td>CEE0EE</td>
<td>3</td>
<td>0462</td>
<td>Instance specific information for the condition token with message number message-number and facility ID facility-id could not be found.</td>
</tr>
</tbody>
</table>

**Usage notes**

- **PL/I consideration:** Conditions with a facility_ID of IBM cannot be used in CEESGL.
- **z/OS UNIX consideration:** In multithread applications, CEESGL affects only the calling thread. Delivery of a CEESGL generated condition is limited to the thread that generated the condition. However, if the condition is a severity 2 or higher, and is not handled by the application, the default action of terminate applies to the enclave, not just the calling thread.
- **COBOL consideration:** When a resume occurs and control resumes to the next instruction that follows the call to CEE3SRP, the COBOL RETURN-CODE special register contains an unpredictable value.

**For more information**

- See [“CEECMI—Store and load message insert data” on page 222](#) for more information about the CEECMI callable service.
- See [z/OS Language Environment Programming Guide](#) for more information about mapping Language Environment condition tokens to POSIX signals.
- See [“ERRCOUNT” on page 31](#) for more information about the ERRCOUNT run-time option.
- To construct your own condition token, see [“CEENCOD—Construct a condition token” on page 407](#).
- For more information about condition handling, see [z/OS Language Environment Programming Guide](#).
- See [“CEEGQDT—Retrieve q_data_token” on page 321](#) for more information about the CEEGQDT callable service.

**Examples**

Figure 200 on page 456 is an example of CEESGL called by C/C++.
/*Module/File Name: EDCSGL */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc,condtok;
    _ENTRY routine;
    _INT4 token,qdata;
    _INT2 c_1,c_2,cond_case,sev,control;
    _CHAR3 facid;
    _INT4 isi;
    /* build the condition token */
    c_1 = 1;
    c_2 = 99;
    cond_case = 1;
    sev = 1;
    control = 0;
    memcpy(facid,"ZZZ",3);
    isi = 0;
    CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
        facid,&isi,&condtok,&fc);
    if (_FBCHECK(fc,CEE000) != 0) {
        printf("CEENCOD failed with message number %d\n",fc.tok_msgno);
        exit(2999);
    }
    /* signal the condition */
    CEESGL(&condtok,&qdata,&fc);
    if (_FBCHECK(fc,CEE000) != 0) {
        printf("CEESGL failed with message number %d\n",fc.tok_msgno);
        exit(2999);
    }
}
/* */
/* */
/* Figure 200. C/C++ Example of CEESGL */

Figure 201 on page 457 is an example of CEESGL called by COBOL.
CEESGL

Figure 201. COBOL Example of CEESGL (Part 1 of 2)
PARA-CBLSGL.
************************************************
** Call CEENCOD with the values assigned above
** to build a condition token "CONDTOK"
** Set CONDTOK to sev = 0, msgno = 1 facid = CEE
************************************************
MOVE 0 TO SEV.
MOVE 1 TO MSGNO.
MOVE 1 TO CASE.
MOVE 0 TO SEV2.
MOVE 1 TO CNTRL.
MOVE "CEE" TO FACID.
MOVE 0 TO ISINFO.

CALL "CEENCOD" USING SEV, MSGNO, CASE,
    SEV2, CNTRL, FACID, ISINFO, CONDTOK, FC.
IF NOT CEE000 of FC THEN
    DISPLAY "CEENCOD failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.

** Call CEESGL to signal the condition with
************************************************
** the condition token and qdata described
** in CONDTOK and QDATA
************************************************
MOVE 0 TO QDATA.
CALL "CEESGL" USING CONDTOK, QDATA, FC.
IF NOT CEE000 of FC THEN
    DISPLAY "CEESGL failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.
GOBACK.

Figure 201. COBOL Example of CEESGL (Part 2 of 2)

Figure 202 on page 459 is an example of CEESGL called by PL/I.
CEESTXF—Transform string characters into collation weights

CEESTXF, which is analogous to the C language function `strcfrm()`, transforms every character in a character string to its unique collation weight. The collation weights are established from the LC_COLLATE category for the locale. CEESTFX also returns the length of the transformed string. CEESTXF is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

```plaintext
*PROCESS MACRO;
/ * Module/File Name: IBMSGL */
/ ******************************************/ *
/ ** Function: CEESGL - Signal a Condition **/
/ ******************************************/
PLISGL: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMMAW;
%INCLUDE CEEIBMCT;

DCL 01 CONDTOK, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

DCL QDATA REAL FIXED BINARY(31,0);

DCL 01 FC, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

/* Give CONDTOK value of condition CEE001 */
ADDR( CONDTOK ) -> CEEIBMCT = CEE001;

/* Signal condition CEE001 with qualifying data */
QDATA = 1;
CALL CEESGL ( CONDTOK, QDATA, FC );
IF FBCHECK( FC, CEE000) THEN DO;
   PUT SKIP LIST( 'Condition CEE001 signalled' );
END;
ELSE DO;
   DISPLAY( 'CEESGL failed with msg '
      || FC.MsgNo );
   STOP;
END;

END PLISGL;

Figure 202. PL/I Example of CEESGL

CEESTXF—Transform string characters into collation weights

CEESTXF, which is analogous to the C language function `strcfrm()`, transforms every character in a character string to its unique collation weight. The collation weights are established from the LC_COLLATE category for the locale. CEESTFX also returns the length of the transformed string. CEESTXF is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.

Chapter 4. Language Environment callable services 459
CEESTXF

Syntax

```c
CEESTXF(-omitted_parm-, -mbstring-, -number-, -txfstring-, -length-, -fc-)
```

- **omitted_parm**
  This parameter is reserved for future expansion and must be omitted. For information on how to code an omitted parm, see "Invoking callable services" on page 112.

- **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/optional)**
  A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following Feedback Code can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE3T1</td>
<td>3</td>
<td>4001</td>
<td>General Failure: Service could not be completed.</td>
</tr>
<tr>
<td>CEE3TF</td>
<td>3</td>
<td>4015</td>
<td>Input Error: The number of characters to be transformed must be greater than zero.</td>
</tr>
</tbody>
</table>

Usage notes

- **PL/I MTF consideration**—CEESETL is not supported in PL/I MTF applications.
- This callable service uses the C/C++ run-time library. The C/C++ library must be installed and accessible even when this service is called from a non-C program.

For more information

- For more information about the `setlocale()` see "COUNTRY" on page 22, "CEE3CTY—Set default country" on page 132, and "CEE3LNG—Set national language" on page 174.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 448.
Examples

Figure 203 is an example of CEESTXF called by COBOL.

```
CBL LIB,QUOTE
*Module/File Name: IGZTSTXF
*************************************************
* Example for callable service CEESTXF *
* COBSTXF - Query current collate category and *
* build input string as function of *
* locale name. *
* Translate string as function of *
* locale. *
* Valid only for COBOL for MVS & VM Release 2 *
* or later. *
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSTXF.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 MBS.
   02 MBS-Length PIC S9(4) BINARY.
   02 MBS-String PIC X(10).
01 TXF.
   02 TXF-Length PIC S9(4) BINARY.
   02 TXF-String PIC X(256).
01 Locale-Name.
   02 LN-Length PIC S9(4) BINARY.
   02 LN-String PIC X(256).
* Use Locale category constants
COPY CEEIGZLC.
01 MBS-Size PIC S9(9) BINARY VALUE 0.
01 TXF-Size PIC S9(9) BINARY VALUE 0.
01 FC.
   02 Condition-Token-Value.
COPY CEEIGZCT.
   03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
   03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
   03 Case-Sev-Ctl PIC X.
   03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
*************************************************
* Call CEEQRYL to retrieve locale name *
*************************************************
CALL "CEEQRYL" USING LC-COLLATE,
   Locale-Name, FC.
*************************************************
* Check feedback code and set input string *
*************************************************
IF Severity = 0
   IF LN-String (1:LN-Length) =
      "Sv-SE.IBM-1047"
   MOVE 10 TO MBS-Length
   MOVE 10 TO MBS-Size
   MOVE "7,123,456."
   TO MBS-String (1:MBS-Length)
```

Figure 203. COBOL Example of CEESTXF (Part 1 of 2)
Figure 203. COBOL Example of CEESTXF (Part 2 of 2)

Figure 204 is an example of CEESTXF called by PL/I.

*PROCESS MACRO;
/*Module/File Name: IBMSTXF
/*Example for callable service CEESTXF
/*Function: Query current collate category and
/*build input string as function of locale name.
/*Translate string as function of locale.

PLISTXF: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants
%INCLUDE CEEIBMLC; /* Locale category constants

Figure 204. PL/I Example of CEESTXF (Part 1 of 2)
CEETDLI

CEETDLI provides an interface to PL/I facilities that operates in IMS. In assembler, COBOL, PL/I, and C/C++, you can also invoke IMS by using the following interfaces:

- In assembler, the ASMTDLI interface
- In COBOL, the CBTLDLI interface
- In PL/I, the PLITDLI interface
- In C/C++, the CTDLI interface, a ctdli() function call

CEETDLI performs essentially the same functions as these interfaces, but it offers some advantages, particularly if you plan to run an ILC application in IMS.
The names CEETDLI, AIBTDLI, ASMTDLI, CBLTDLI, CTDLI, and PLITDLI are all interpreted as IMS interfaces. If you are currently using them in any other way in your application, you must change them. The CEETDLI interface supports calls to IMS that use an application interface block (AIB) or a program communication block (PCB).

**Syntax**

```
//SM590000/SM590000
CEETDLI(
  parmcount,
  function,
  args
)/SM590000/SM630000
```

- **parmcount**
  A fullword integer specifying the total number of arguments for the CEETDLI call. This option usually is not needed and can be omitted; it is supported for compatibility with earlier interface modules.

- **function**
  The IMS function that you want to perform. The possible values for this field are defined by IMS, not Language Environment.

- **args**
  Arguments that you pass to IMS. You cannot pass run-time options as CEETDLI arguments. You cannot alter the settings of run-time options when invoking IMS facilities. The order and meaning of the arguments is defined by IMS, not Language Environment.

**Usage notes**

- CEETDLI is not supported with CICS.
- z/OS UNIX considerations—IMS supports only applications that use the POSIX(ON) run-time option from a single thread. Calls to z/OS UNIX threading functions are restricted under IMS. See z/OS XL C/C++ Programming Guide for a list of restrictions on running IMS and z/OS UNIX.

**For more information**

- For more information about AIB and a complete description of all available IMS functions and argument parameters you can specify in CEETDLI, see IMS Application Programming Guide listed in the Bibliography of this document.
- For more information about CEETDLI in the context of other DLI interfaces, and in the context of IMS condition handling, see z/OS Language Environment Programming Guide.

**Examples**

Figure 205 on page 465 is an example of CEETDLI called by C.
Figure 206 is an example of CEETDLI called by COBOL.

```c
/*Module/File Name: EDCMRCR */
#pragma runopts(env(IMS),plist(IMS))
#include <ims.h>
#include <leawi.h>
#define io_pcb ((__pcblist??(0??))
/
/* ------------------------------------------------ */
/* */
/* Function: Use CEETDLI - interface to IMS */
/* from C. */
/* */
/* In this example, a call is made to CEETDLI */
/* to interface to IMS for an IMS service. */
/* */
/* ------------------------------------------------ */
/* ENTRY POINT */
/* ------------------------------------------------ */
main() {
    static char func_GU??(4??) = "GU ";
    char msg_seg_io_area??(32??);,
    CEETDLI(func_GU,io_pcb,msg_seg_io_area);
}
```

Figure 206. C Example of CEETDLI

Figure 206 is an example of CEETDLI called by COBOL.

```cobol
+Module/File Name: IGZTDLI
******************************************************************************
** **
** Function: Use CEETDLI - interface to IMS **
** from COBOL. **
** **
** In this example, a call is made to CEETDLI **
** to interface to IMS for an IMS service. **
** **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL2IMS.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 FUNC-GU PIC X(4) VALUE "GU ".
01 MSG-SEG-IO-AREA PIC X(32).
LINKAGE SECTION.
01 IO-PCH PIC X(48).
** ENTRY POINT:
PROCEDURE DIVISION USING IO-PCH.
    CALL "CEETDLI" USING FUNC-GU,IO-PCH,
            MSG-SEG-IO-AREA.
GOBACK.
```

Figure 206. COBOL Example of CEETDLI

Figure 206 is an example of CEETDLI called by COBOL.

```
```
CEETEST—Invoke Debug Tool

CEETEST invokes a debug service such as Debug Tool, which is supplied with z/OS.

Debug Tool supports debugging of Language Environment, except for some noted restrictions. For more information about Debug Tool for z/OS, see [http://www.ibm.com/software/awdtools/debugtool/](http://www.ibm.com/software/awdtools/debugtool/) If you want to invoke another interactive debug service, refer to the appropriate user’s guide.

**Syntax**

```
>CEETEST(string_of_commands, fc)
```

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 Feedback Code can result from this service:
## Usage notes

- z/OS UNIX considerations—CEETEST applies to the enclave. All threads in the enclave can access debugger information.

## For more information

- If you are using CEETEST to invoke Debug Tool and need more information about how to create a Debug Tool command list, refer to [http://www.ibm.com/software/awdtools/debugtool/](http://www.ibm.com/software/awdtools/debugtool/).

## Examples

**Figure 208** is an example of CEETEST called by C/C++.

```
/*Module/File Name: EDCTEST */
#include <string.h>
#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main (void) {
  int x,y,z;
  _VSTRING commands;
  _FEEDBACK fc;
  strncpy(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 */
  /* */
  . */
}
```

**Figure 209** on page 468 is an example of CEETEST called by COBOL.
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(first call) failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.

Figure 209. COBOL Example of CEETEST (Part 1 of 2)
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.

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 .
   GORACK.
END PROGRAM SUBRTN.
END PROGRAM IBCT002.

Figure 209. COBOL Example of CEETEST (Part 2 of 2)

Figure 210 on page 470 is an example of CEETEST called by PL/I.
**CEEUTC**—Get Coordinated Universal Time

CEEUTC is an alias of CEEGMT. See “CEEGMT—Get current Greenwich Mean Time” on page 309 for more information.
Chapter 5. Bit manipulation routines

This section lists the Language Environment bit manipulation routines. In this section, bits are numbered from right to left, starting from 0.

CEESICLR—Bit clear

Purpose
CEESICLR returns a copy of its parm1 input, but with one bit selectively set to 0.

Syntax

```
CEESICLR (parm1, parm2, fc, result)
```

parm1 (input)
The first input to the Bit Clear routine. The input can be any 32-bit integer.

parm2 (input)
The second input to the Bit Clear routine. The parm2 value is a 32-bit integer in the range between 0 and 31, inclusive.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1VC</td>
<td>2</td>
<td>2028</td>
<td>The value of the second argument was outside the valid range range in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
The result of the Bit Clear routine. The output is a copy of parm1, but with the bit numbered parm2 (counting from the right) set to 0.

CEESISET—Bit set

Purpose
CEESISET returns a copy of its parm1 input, but with one bit selectively set to 1.

Syntax

```
CEESISET (parm1, parm2, fc, result)
```
CEESISET—Bit set

parm1 (input)
   The first input to the Bit Set routine. The input can be any 32-bit integer.

parm2 (input)
   The second input to the Bit Set routine. The parm2 value is a 32-bit integer in
   the range between 0 and 31, inclusive.

fc (output)
   A 12-byte feedback code, optional in some languages, that indicates the result
   of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1VC</td>
<td>2</td>
<td>2028</td>
<td>The value of the second argument was outside the valid range in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
   The result of the Bit Set routine. The output is a copy of parm1, but with the bit
   numbered parm2 (counting from the right) set to 1.

CEESISHF—Bit shift

Purpose

CEESISHF returns a copy of its parm1 input right- or left-shifted by the number of
bits indicated by parm2.

Syntax

CEESISHF (parm1, parm2, fc, result)

parm1 (input)
   The first input to the Bit Shift routine. The input can be any 32-bit integer.

parm2 (input)
   The second input to the Bit Shift routine. The parm2 value is a 32-bit integer in
   the range between -32 and 32, inclusive.

fc (output)
   A 12-byte feedback code, optional in some languages, that indicates the result
   of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1VC</td>
<td>2</td>
<td>2028</td>
<td>The value of the second argument was outside the valid range in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
   The result of the Bit Shift routine. The output is a 32-bit integer whose value
   depends upon the value of parm2; in either case, vacated bits are set to 0:
   
   • If parm2 is greater than or equal to 0, result is a copy of parm1 shifted left by
     parm2 bits.
If \(parm2\) is less than 0, \(result\) is a copy of \(parm1\) shifted right by \(|parm2|\) bits.

---

**CEESITST—Bit test**

**Purpose**

CEESITST selectively tests a bit in its \(parm1\) input to determine if the bit is on.

**Syntax**

```
CEESITST(_parm1_, _parm2_, _fc_, _result_)
```

- **parm1** (*input*)
  The first input to the Bit Test routine. The input can be any 32-bit integer.

- **parm2** (*input*)
  The second input to the Bit Test routine. The \(parm2\) value is a 32-bit integer in the range between 0 and 31, inclusive.

- **fc** (*output*)
  A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1VC</td>
<td>2</td>
<td>2028</td>
<td>The value of the second argument was outside the valid range (range) in math routine (routine-name).</td>
</tr>
</tbody>
</table>

- **result** (*output*)
  The result of the Bit Test routine. The output is a 32-bit integer with one of the following values; bits are counted from the right:

  1   if bit number \(parm2\) in \(parm1\) is 1
  0   if bit number \(parm2\) in \(parm1\) is 0
CEESITST—Bit test
Chapter 6. Language Environment math services

Language Environment math services provide standard math computations. You can call them from Language Environment-conforming languages or from assembler routines by using the call interface (defined below and shown throughout this chapter), or the syntax specific to the HLL of your application.

Math services do not depend on enclave-level resources. You can invoke them from any thread.

If your application uses extended-precision arithmetic and runs on a 370-mode machine under VM, you must specify the TRAP(ON) run-time option (the default) and add the CMSLIB TXTLIB with the GLOBAL TXTLIB command.

Call interface to math services

The syntax for math services has two forms, depending on how many input parameters the routine requires. The first four letters of the math services are always CEES. The fifth character is x, which you replace according to the parameter types listed in "Parameter types: parm1 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

───CEESxABS──{parm1, ‹fc›, ‹result›}

Two Parameters

───CEESxDIM──{parm1, parm2, ‹fc›, ‹result›}

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. This parameter type consists of a real part and an imaginary part, each of which is a 32-bit single floating-point number.
E  64-bit double floating-point complex number. This parameter type consists of a real part and an imaginary part, each of which is a 64-bit double floating-point number.

R  128-bit extended floating-point complex number. This parameter type consists of a real part and an imaginary part, each of which is a 128-bit extended floating-point number.

Language Environment math services expect normalized input.

In the routines described in this chapter, the output range for complex-valued functions can be determined from the input range. For functions of complex variables, the image of the input is generally a non-rectangular shape. For this reason, the output range is not provided.

Feedback code parameter (fc)
The fc value is a feedback code that indicates the result of the math service. If you specify fc as an argument, feedback information in the form of a condition token is returned to the calling routine. The condition token indicates whether the routine completed successfully or whether a condition was encountered while the routine was running. If you do not specify fc as an argument and the requested service does not successfully complete, the condition is signaled. Math services call other services that might generate feedback codes.

Language-specific built-in math services
C/C++, COBOL, Fortran, and PL/I offer built-in math services that you can also use under Language Environment. For a description of these functions, see one of the language references listed in the "Bibliography" on page 533.
## Calls to math services from different languages

Table 28 shows sample calls to the Language Environment math service CEESLOG—logarithm base e, as made from C/C++, COBOL, and PL/I. For more examples, see “Examples of math services” on page 509.

### Table 28. Examples of calls to CEESLOG

<table>
<thead>
<tr>
<th>Called from</th>
<th>Code example</th>
</tr>
</thead>
</table>
| C/C++       | `#include <leawi.h>`<br>`#include <string.h>`<br>`#include <stdio.h>`<br><br>`int main (void) {<br>  float int1, intr;<br>  _FEEDBACK fc;<br>  #define SUCCESS "\0\0\0\0"
  int1 = 39;<br>  CEESSLOG(&int1,&fc,&intr);<br>  if (memcmp(&fc,SUCCESS,4) != 0) {<br>    printf("CEESSLOG failed with message number %d\n",
      fc.tok_msgno);
    exit(2999);<br>  }
  printf("Log base e of %f is %f\n",int1,intr);<br>}
| COBOL       | 77 ARG1RS COMP-1.<br> 77 FBCODE PIC X(12).<br> 77 RESLRS COMP-1.<br><br>    CALL "CEESSLOG" USING ARG1RS , FBCODE , RESLRS.<br> |<br>PL/I        | DCL ARG1 RESULT REAL FLOAT DEC (6);<br> DCL FC CHARACTER (12);<br><br>    CALL CEESSLOG (ARG1, FC, RESULT) |}

### Math services

This section describes the Language Environment math services. It also provides examples of how various services are called by different programming languages.

#### CEESxABS—Absolute value

CEESxABS returns the absolute value of the parameter by using the equation:

\[ result = |parm1| \]

The following routines are provided for the various data types supported:

- **CEESIABS** 32-bit binary integer
**CEESxABS**

**Syntax**

```
-> CEESxABS(—parm1—, —fc—, —result—)
```

---

**parm1 (input)**

The input to the absolute value routine. The input range is not restricted.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>routine-name.</td>
</tr>
</tbody>
</table>

**result (output)**

The result of the absolute value routine. The output range is the non-negative numbers:

\[ \leq \Omega \]

Omega varies depending on the precision of `parm1`:

- For single-precision routines, \( \Omega = 16^{63}(1 - 16^{-6}) \)
- For double-precision routines, \( \Omega = 16^{63}(1 - 16^{-14}) \)
- For extended-precision routines, \( \Omega = 16^{63}(1 - 16^{-28}) \)

**CEESxACS—Arccosine**

CEESxACS returns the arccosine of the parameter by using the equation: `result = \text{arccos}(\text{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
CEESxACS

Syntax

```c
CEESxACS((parm1, fc, result))
```

**parm1 (input)**

The input to the arccosine routine. The input range is:

|parm1| \leq 1

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1V0</td>
<td>2</td>
<td>2016</td>
<td>The absolute value of the parameter was greater than limit in math routine routine-name.</td>
</tr>
</tbody>
</table>

**result (output)**

The result, in radians, of the arccosine routine.

| 0 \leq result \leq \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

```c
CEESxASN((parm1, fc, result))
```

**parm1 (input)**

The input to the arcsine routine. The input range is:
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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1V0</td>
<td>2</td>
<td>2016</td>
<td>The absolute value of the parameter was greater than limit in math routine routine-name.</td>
</tr>
</tbody>
</table>

The result, in radians, of the arcsine routine. The output range is:

\[ |result| < \pi/2 \]

**CEESxATH—Hyperbolic arctangent**

CEESxATH returns the hyperbolic arctangent of the parameter by using the following equation:

\[ result = \tanh^{-1}(parm1) \]

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  128-bit extended floating-point number
- CEESTATH  32-bit single floating-point complex number
- CEESETH  64-bit double floating-point complex number
- CEESRATH  128-bit extended floating-point complex number

**Syntax**

\[
\text{---CEESxATH---}(\text{-parm1-}, \text{-fc-}, \text{-result-})
\]

**parm1 (input)**

The input to the hyperbolic arctangent routine. The input range for real variables is:

\[ |parm1| \leq 1 \]
For complex variables, \( \text{parm1} \) cannot be equal to 1 or \(-1\).

\( f_c \) (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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1V1</td>
<td>2</td>
<td>2017</td>
<td>The absolute value of the argument was greater than or equal to ( \text{limit} ) in math routine ( \text{routine-name} ).</td>
</tr>
<tr>
<td>CEE1V6</td>
<td>2</td>
<td>2022</td>
<td>The value of the argument was plus or minus ( \text{limit} ) in math routine ( \text{routine-name} ).</td>
</tr>
</tbody>
</table>

\( \text{result} \) (output)

The result of the hyperbolic arctangent routine. The output range for functions of real variables is:

\[ |\text{result}| \leq \Omega \]

Omega varies, depending on the precision of \( \text{parm1} \):

- 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: \( \text{result} = \arctan(\text{parm1}) \)

The following routines are provided for the various data types supported:

- **CEESATN** 32-bit single floating-point number
- **CEESDATN** 64-bit double floating-point number
- **CEESQATN** 128-bit extended floating-point number
- **CEESTATN** 32-bit single floating-point complex number
- **CEESEATN** 64-bit double floating-point complex number
- **CEESRATN** 128-bit extended floating-point complex number

**Syntax**

\[ \text{CEESxATN}(-\text{parm1}, -\text{fc}, -\text{result}) \]
parml (input)
The input to the arctangent routine. The input range for real variables is not restricted. The input range of complex variables in parml 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1V6</td>
<td>2</td>
<td>2022</td>
<td>The value of the parameter was plus or minus limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
The result, in radians, of the arctangent routine. The output range for functions of real variables is:

\[ |\text{result}| < \pi/2 \]

**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, parml) with a range from:

\[-\pi \text{ to } \pi, \text{ with } \pi,\]

For example, if parml and parm2 are positive, then result = arctan (parml/parm2).

The following routines are provided for the various data types supported:

- **CEESSAT2** 32-bit single floating-point number
- **CEESDAT2** 64-bit double floating-point number
- **CEESQAT2** 128-bit extended floating-point number

**Syntax**

```
// CEESxAT2 paralysis paralysis fc result
```
parm1 (input)
The first input to the arctangent2 routine. The input range of parm1 cannot
equal 0 if parm2 equals 0.

parm2 (input)
The second parameter to the arctangent2 routine. The input range of parm2
cannot equal 0 if parm1 equals 0.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result
of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UU</td>
<td>2</td>
<td>2014</td>
<td>Both parameters were equal to limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine routine-name.</td>
</tr>
</tbody>
</table>

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

```c
CEESxCJG((parm1, fc, result))
```

parm1 (input)
The input to the math service. Any representable complex number can be used
as input.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result
of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

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

```c
CEESxCOS(parm1, fc, result)
```

**parm1** (input)
The type is determined by the fifth character of the service name. The input range for real variables varies:

<table>
<thead>
<tr>
<th>Type</th>
<th>Input Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single floating-point numbers</td>
<td>(</td>
</tr>
<tr>
<td>Double floating-point numbers</td>
<td>(</td>
</tr>
<tr>
<td>Extended floating-point numbers</td>
<td>(</td>
</tr>
</tbody>
</table>

For complex functions, the input range differs for the imaginary and real parts of the input.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Input Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imaginary part: single floating-point numbers</td>
<td>(</td>
</tr>
<tr>
<td>Imaginary part: double floating-point numbers</td>
<td>(</td>
</tr>
<tr>
<td>Imaginary part: extended floating-point numbers</td>
<td>(</td>
</tr>
<tr>
<td>Real part: single floating-point complex numbers</td>
<td>(</td>
</tr>
<tr>
<td>Real part: double floating-point complex numbers</td>
<td>(</td>
</tr>
<tr>
<td>Real part: extended floating-point complex numbers</td>
<td>(</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UT</td>
<td>2</td>
<td>2013</td>
<td>The absolute value of the imaginary part of the parameter was greater than limit in math routine routine-name.</td>
</tr>
</tbody>
</table>
### CEESxCOS

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE1V1</td>
<td>2</td>
<td>2017</td>
<td>The absolute value of the parameter was greater than or equal to limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1V3</td>
<td>2</td>
<td>2019</td>
<td>The absolute value of the real part of the parameter was greater than or equal to limit in math routine routine-name.</td>
</tr>
</tbody>
</table>

**result (output)**
The result of the cosine routine. The output range for functions of real variables is:

\[ |result| \leq 1 \]

#### CEESxCOSH—Hyperbolic cosine

CEESxCOSH 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
- **CEESDCHS** 64-bit double floating-point number
- **CEESQCHS** 128-bit extended floating-point number
- **CEESTCSh** 32-bit single floating-point complex number
- **CEESECCHS** 64-bit double floating-point complex number
- **CEESRCSCH** 128-bit extended floating-point complex number

**Syntax**

\[
\text{CEESxCSH}(\text{parm1}, f_{c}, \text{result})
\]

**parm1 (input)**
The input to the hyperbolic cosine routine. The input range varies, depending on the function and type of number:

| For functions of real variables: | \( |\text{parm1}| < 175.366 \) |
|----------------------------------|-------------------------|
| For the real part of complex numbers: | \( |\text{Re}(\text{parm1})| < 174.673 \) |
| For the imaginary part of complex numbers: single floating-point complex numbers: | \( |\text{Im}(\text{parm1})| < (2^{28} \cdot 11) \) |
| For the imaginary part of complex numbers: double floating-point complex numbers: | \( |\text{Im}(\text{parm1})| < (2^{50} \cdot 11) \) |
| For the imaginary part of complex numbers: extended floating-point complex numbers: | \( |\text{Im}(\text{parm1})| < 2^{100} \) |
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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1V0</td>
<td>2</td>
<td>2016</td>
<td>The absolute value of the parameter was greater than $\text{limit}$ in math routine $\text{routine-name}$.</td>
</tr>
</tbody>
</table>

result **(output)**
The result of the hyperbolic cosine routine. The output range for functions of real variables varies:

| For single-precision routines: | $1 \leq \text{result} \leq \Omega$ |
| For double-precision routines: | $\Omega = 16^{63}(1 - 16^{16})$ |
|                                | $16^{63}(1 - 16^{-14})$           |
| For extended-precision routines: | $16^{63}(1 - 16^{-28})$          |

**CEESxCSTN—Cotangent**
CEESxCSTN returns the cotangent of the parameter by using the equation: $\text{result} = \cot(\text{parm1})$

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**

```c
>>>CEESxCSTN(--parm1--,--fc--,--result--) <<<
```

**parm1 (input)**
The input, in radians, into the cotangent routine. The input range varies, depending on the value of $\text{parm1}$:

| If $\text{parm1}$ is a 32-bit single floating-point number: | $|\text{parm1}| < (2^{16} \cdot \Pi)$ |
| If $\text{parm1}$ is a 64-bit double floating-point number: | $|\text{parm1}| < (2^{20} \cdot \Pi)$ |
| If $\text{parm1}$ is a 128-bit extended floating-point number: | $|\text{parm1}| < 2^{100}$ |
If this is an extended floating-point number, this argument cannot approach a multiple of pi \((\pi)\). 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UI</td>
<td>2</td>
<td>2002</td>
<td>The parameter value was too close to one of the singularities (plus or minus (\pi/2), plus or minus (3\pi/2), for the tangent; or plus or minus (\pi), plus or minus (2\pi), for the cotangent) in math routine <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1V1</td>
<td>2</td>
<td>2017</td>
<td>The absolute value of the parameter was greater than or equal to (limit) in math routine <code>routine-name</code>.</td>
</tr>
</tbody>
</table>

\( result \) **(output)**

The result of the cotangent routine. The output range is:

\[
|result| \leq \Omega
\]

For single-precision routines, \( \Omega = 16^{63}(1 - 16^{-6}) \)
For double-precision routines, \( \Omega = 16^{63}(1 - 16^{-16}) \)
For extended-precision routines, \( \Omega = 16^{63}(1 - 16^{-28}) \)

The output range is:

\[ result \leq \Omega \]

**CEESxDIM—Positive difference**

CEESxDIM returns the positive difference between two numbers by using one of the following equations:

<table>
<thead>
<tr>
<th>If (parm1 &gt; parm2)</th>
<th>Then, (result = parm1 - parm2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>If (parm1 \leq parm2)</td>
<td>Then, (result = 0)</td>
</tr>
</tbody>
</table>

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
CEESxDIM

Syntax

```plaintext
CEESxDIM((parm1, parm2, fc, result))
```

**parm1 (input)**
The first input to the positive difference routine. The input range is not restricted.

**parm2 (input)**
The second parameter to the positive difference routine. The input range is not restricted.

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**result (output)**
The result of the positive difference routine. The output range is the non-negative numbers.

\[ \Omega \]

For single-precision routines, \( \Omega = 16^{63}(1 - 16^{-6}) \)
For double-precision routines, \( \Omega = 16^{63}(1 - 16^{-16}) \)
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:

\[
\text{result} = \frac{\text{parm1}}{\text{parm2}}
\]

The following routines are provided for the various data types supported:

- **CEESTDVD** - 32-bit single floating-point complex number
- **CEESEDVD** - 64-bit double floating-point complex number
- **CEESRDVD** - 128-bit extended floating-point complex number
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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

result (output)
The result of the floating-point complex divide routine.

**CEESxERC—Error function complement**
CEESxERC calculates the error function complement by using the equation:

\[ result = \left(1 - \frac{2}{\sqrt{\pi}} \int_{0}^{\text{parm1}} e^{-u^2} du \right) \]

The following routines are provided for the various data types supported:

- **CEESSERC**: 32-bit single floating-point number
- **CEESDERC**: 64-bit double floating-point number
- **CEESQERC**: 128-bit extended floating-point number

**Syntax**

```
CEESxERC( parm1, fc, result )
```

parm1 (input)
The input to the error function complement routine. The input range is not restricted.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

result (output)
The result of the error function complement routine. The output range is: \( 0 < \text{result} < 2 \).
CEESxERF—Error function

CEESxERF calculates the error function by using the equation:

\[ \text{result} = \frac{2}{\sqrt{\pi}} \int_{0}^{\infty} e^{-u^2} du \]

The following routines are provided for the various data types supported:

- **CEESSERF**: 32-bit single floating-point number
- **CEESDERF**: 64-bit double floating-point number
- **CEESQERF**: 128-bit extended floating-point number

**Syntax**

```
CEESxERF(parm1, fc, result)
```

**parm1 (input)**

The input to the error function. The input range is not restricted.

**fc (output)**

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**result (output)**

The result of the error function. The output range is:

\[ |\text{result}| \leq 1 \]

CEESxEXP—Exponential base e

CEESxEXP calculates the mathematical function of e raised to a power by using the equation:

\[ \text{result} = e^{\text{parm1}} \]

The following routines are provided for the various data types supported:

- **CEESSEXP**: 32-bit single floating-point number
- **CEESDEXP**: 64-bit double floating-point number
- **CEESQEXP**: 128-bit extended floating-point number
- **CEESTEXP**: 32-bit single floating-point complex number
- **CEESEEXP**: 64-bit double floating-point complex number
CEESREXP 128-bit extended floating-point complex number

Syntax

```
CEESxEXP parm1, fc, result
```

**parm1 (input)**
The input to the exponential base e routine. The input range varies, depending on the type of function and number:

<table>
<thead>
<tr>
<th>Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>For functions of real variables:</td>
<td>(</td>
</tr>
<tr>
<td>For the real part of complex numbers:</td>
<td>(</td>
</tr>
<tr>
<td>For the imaginary part: single floating-point complex numbers:</td>
<td>(</td>
</tr>
<tr>
<td>For the imaginary part: double floating-point complex numbers:</td>
<td>(</td>
</tr>
<tr>
<td>For the imaginary part: extended floating-point complex numbers:</td>
<td>(</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UP</td>
<td>2</td>
<td>2009</td>
<td>The value of the real part of the parameter was greater than limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1UR</td>
<td>2</td>
<td>2011</td>
<td>The parameter was greater than limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1UT</td>
<td>2</td>
<td>2013</td>
<td>The absolute value of the imaginary part of the parameter was greater than limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1UV</td>
<td>2</td>
<td>2015</td>
<td>The absolute value of the imaginary part of the parameter was greater than or equal to limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine routine-name.</td>
</tr>
</tbody>
</table>

**result (output)**
The result of the exponential base e routine. The output range for functions of real variables is:
CEESxEXP

0 < result ≤ Ω
For single-precision routines, Ω = 16^{63}(1 - 16^{-6})
For double-precision routines, Ω = 16^{63}(1 - 16^{-14})
For extended-precision routines, Ω = 16^{63}(1 - 16^{-28})

CEESxGMA—Gamma function
CEESxGMA performs the mathematical gamma function by using the equation:

result = \int_{0}^{\infty} u_{param1}^{-1} e^{-u} du

The following routines are provided for the various data types supported:

CEESGMA 32-bit single floating-point number
CEESDGMA 64-bit double floating-point number

Syntax

```c
CEESxGMA (parm1, fc, result)
```

parm1 (input)
The input to the gamma function. The input range is:

2^{252} < parm1 < 57.5744

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UL</td>
<td>2</td>
<td>2005</td>
<td>The value of the parameter was outside the valid range range in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
The result of the gamma function. The output range varies, depending on the type of routine:

For single-precision routines:

0.88560 ≤ result ≤ Ω

Ω = 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

\[ \text{result} = v, \text{ where } \text{parm1} = u + vi. \]

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**

```c
CEESxIMG (parm1, fc, result)
```

*parm1* *(input)*

The input to the math service. Any complex number can be used as input.

*fc* *(output)*

A 12-byte feedback code passed by reference indicating the result of the service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

*result* *(output)*

The result of the math service is:

\[ |\text{result}| \leq \Omega \]

For single-precision routines, \( \Omega = 16^{63}(1-16^{-6}) \)

For double-precision routines, \( \Omega = 16^{63}(1-16^{-14}) \)

For extended-precision routines, \( \Omega = 16^{63}(1-16^{-28}) \)

**CEESxINT—Truncation**

CEESxINT returns the truncated value of the parameter by using the equation, where \( m \) is the greatest integer satisfying the relationship:

\[ \text{result} = (\text{sign of parm1}) \cdot n, \text{ where } n = |\text{parm1}| \]

\[ |\text{parm1}| = |m| \]

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

```
CEESxINT( parm1, fc, result )
```

parm1 (**input**)  
The input to the truncation routine. The input range is not restricted.

fc (**output**)  
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

result (**output**)  
The result of the truncation routine. The output range is:

\[ |result| \leq \Omega \]

For single-precision routines, \( \Omega = 16^{63}(1 - 16^{-6}) \)
For double-precision routines, \( \Omega = 16^{63}(1 - 16^{-15}) \)
For extended-precision routines, \( \Omega = 16^{63}(1 - 16^{-28}) \)

### CEESxLGM—Log gamma

CEESxLGM performs the mathematical function of log gamma by using either of the following equations:

\[
result = \log_e \Gamma(\text{parm1})
\]

\[
result = \log \int_{0}^{\infty} u^{\text{parm1} - 1} e^{-u} \, du
\]

The following routines are provided for the various data types supported:

**CEESSLGM** 32-bit single floating-point number
**CEESDLGM** 64-bit double floating-point number
Syntax

```latex
\texttt{CEE\textunderscore x\textunderscore LGM\{\texttt{-param1\},\texttt{-fc\},\texttt{-result}\}}
```

**parm1 (input)**

The input to the log gamma routine. The input range is:

\( \text{parm1} < 4.2913 \cdot 10^3 \)

**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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UL</td>
<td>2</td>
<td>2005</td>
<td>The value of the parameter was outside the valid range in math routine ( \text{routine-name} ).</td>
</tr>
</tbody>
</table>

**result (output)**

The result of the log gamma routine. The output range varies:

- For single-precision routines: \(-0.12149 \leq \text{result} \leq \Omega\)
- For double-precision routines: \(\Omega = 16^{63} (1-16^6)\) \(\Omega = 16^{63} (1-16^{14})\)

**CEES\textunderscore x\textunderscore LG1—Logarithm base 10**

CEES\textunderscore x\textunderscore LG1 returns the logarithm base 10 of the input parameter by using the equation:

\( \text{result} = \log_{10} \text{parm1} \)

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
CEESxLG1

Syntax

```
CEESxLG1 parm1, fc, result
```

parm1 (input)
The input to the log base 10 routine. The input range is: \(parm1 > 0\).

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1US</td>
<td>2</td>
<td>2012</td>
<td>The parameter was less than or equal to (limit) in math routine (routine-name).</td>
</tr>
</tbody>
</table>

result (output)
The result of the log base 10 routine. The output range is: \(result\) is greater than or equal to -78.268 and is less than or equal to 75.859 (that is, \(-78.268 \leq result \leq 75.859\)).

CEESxLG2—Logarithm base 2

CEESxLG2 performs the mathematical function logarithm base 2 by using the equation:

\[
result = \log_2 parm1
\]

The following routines are provided for the various data types supported:

- CEESSLG2 32-bit single floating-point number
- CEESDLG2 64-bit double floating-point number
- CEESQLG2 128-bit extended floating-point number

Syntax

```
CEESxLG2 parm1, fc, result
```

parm1 (input)
The input to the log base 2 routine. The input range is: \(parm1 > 0\).

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>
result (output)
The result of the log base 2 routine. The output range is: result is greater than or equal to -260 and is less than or equal to 252 (that is, -260 \leq result \leq 252).

CEESxLOG—Logarithm base e
CEESxLOG performs the mathematical function logarithm base e by using the equation:

\[ result = \log_e \text{parm1} \]

The following routines are provided for the various data types supported:

- CEESLOG 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

```plaintext
>>> CEESxLOG({parml,-,fc,-,result-})
```

parml (input)
The input to the log base e routine. The input range varies:

- For real numbers: \( parml > 0 \)
- For complex numbers: \( parml \) is not equal to 0 (that is, \( parml \neq 0 \))

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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1US</td>
<td>2</td>
<td>2012</td>
<td>The parameter was less than or equal to limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1V2</td>
<td>2</td>
<td>2018</td>
<td>The real and imaginary parts of the parameter were equal to limit in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
The result of the log base e routine. The output range for functions of real
variables is: result is greater than or equal to -180.218 and is less than or equal to \( \leq 174.673 \) (that is, \(-180.218 \leq \text{result} \leq 174.673\)).

**CEESxMLT—Floating-point complex multiply**

CEESxMLT performs the mathematical function floating-point complex multiply by using the equation:

\[
\text{result} = \text{parm1} \cdot \text{parm2}
\]

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**

```c
CEESxMLT (parm1, parm2, fc, result)
```

*parm1 (input)*

The first input to the math service. Any representable complex number can be used as input.

*parm2 (input)*

The second parameter to the math service. Any representable complex number can be used as input.

*fc (output)*

A 12-byte feedback code passed by reference indicating the result of the service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

*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:

\[
\text{result} = \text{parm1} \pmod{\text{parm2}}
\]

The expression \( \text{parm1} \pmod{\text{parm2}} \) is defined as follows, with the brackets indicating an integer part:

\[
\text{parm1} - \left\lfloor\frac{\text{parm1}}{\text{parm2}}\right\rfloor \cdot \text{parm2}
\]
CEESxMOD

That is, the largest integer whose magnitude does not exceed the magnitude of \( \text{parm1}/\text{parm2} \) is used.

The sign of the integer is the same as the sign of \( \text{parm1} \cdot \text{parm2} \).

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**: 128-bit extended floating-point number

**Syntax**

\[
\text{CEESxMOD} (\text{parm1}, \text{parm2}, \text{fc}, \text{result})
\]

**parm1** *(input)*

The first parameter to the modular arithmetic routine. The input range is not restricted.

**parm2** *(input)*

The second parameter to the modular arithmetic routine. The input range is: \( \text{parm2} \) is not equal to 0 (that is, \( \text{parm2} \neq 0 \))

If \( \text{parm2} = 0 \), 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**result** *(output)*

The result of the mod routine. The output range is:

\[|\text{result}| \leq \Omega\]

For single-precision routines, \( \Omega = 16^{63}(1 - 16^6) \)

For double-precision routines, \( \Omega = 16^{63}(1 - 16^{16}) \)

For extended-precision routines, \( \Omega = 16^{63}(1 - 16^{28}) \)

**CEESxNIN—Nearest integer**

CEESxNIN performs the mathematical function nearest integer by using the equation:
CEESxNIN

\[
result = (\text{sign of } \text{parm1}) \cdot n
\]

If \( \text{parm1} \geq 0 \), then \( n = \lfloor \text{parm1} + 0.5 \rfloor \)

If \( \text{parm1} < 0 \), then \( n = \lfloor \text{parm1} - 0.5 \rfloor \)

\( n = |m| \), where \( m \) is the greatest integer satisfying the relationship \( |m| \leq |\text{parm1} + 0.5| \)

or \( |m| \leq |\text{parm1} - 0.5| \), 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**

```ce
>>> CEESxNIN((parm1, fc, result))
```

**parm1** (input)

The input to the nearest integer routine. The input range is not restricted.

**fc** (output)

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**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:

\[
result = (\text{sign of } \text{parm1}) \cdot v
\]

If \( \text{parm1} \geq 0 \), then \( v = \lfloor \text{parm1} + 0.5 \rfloor \). If \( \text{parm1} < 0 \), then \( v = \lfloor \text{parm1} - 0.5 \rfloor \).

\( v = |m| \), where \( m \) is the greatest integer satisfying the relationship \( |m| \leq |\text{parm1} + 0.5| \)

or \( |m| \leq |\text{parm1} - 0.5| \), respectively.

and the resulting \( v \) is expressed as a floating-point number.

The following routines are provided for the various data types supported:

- **CEESSNWN**: 32-bit single floating-point number
- **CEESDNWN**: 64-bit double floating-point number
**Syntax**

```
>>> CEESxNWN( parm1, fc, result )
```

**parm1** *(input)*

The input to the nearest whole number routine. The input range is not restricted.

**fc** *(output)*

A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**result** *(output)*

The result of the nearest whole number routine. The output range varies:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>For single-precision routines:</td>
<td>result | \leq \Omega</td>
</tr>
<tr>
<td></td>
<td>\Omega = 16^{63} (1 \text{-} 16^{6})</td>
</tr>
<tr>
<td>For double-precision routines:</td>
<td>\Omega = 16^{63} (1 \text{-} 16^{14})</td>
</tr>
</tbody>
</table>

**CEESxSGN—Transfer of sign**

CEESxSGN performs the mathematical function transfer of sign by using *either* of the two equations:

\[
result = |parm1| \text{ if } parm2 \geq 0 \text{ or } result = -|parm1| \text{ if } parm2 < 0.
\]

The following routines are provided for the various data types supported:

- **CEESISGN** 32-bit binary integer
- **CEESSSGN** 32-bit single floating-point number
- **CEESDSGN** 64-bit double floating-point number
- **CEESQSGN** 128-bit extended floating-point number

**Syntax**

```
>>> CEESxSGN( parm1, parm2, fc, result )
```

**parm1** *(input)*

The first input to the transfer of sign routine. The input range is not restricted.

**parm2** *(input)*

The second input 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.
CEESxSGN

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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

result (output)
The result of the transfer of sign routine. The output range is:

\[ |\text{result}| \leq \Omega \]

- For single-precision routines, \( \Omega = 16^{63}(1 - 16^{-6}) \)
- For double-precision routines, \( \Omega = 16^{63}(1 - 16^{-16}) \)
- 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
- CEEDDSIN 64-bit double floating-point number
- CEESQQSIN 128-bit extended floating-point number
- CEESTSIN 32-bit single floating-point complex number
- CEESESIN 64-bit double floating-point complex number
- CEESRSIN 128-bit extended floating-point complex number

Syntax

```
-----CEESxSIN---(parm1--,fc--,result--)---
```

parm1 (input)
The input, in radians, to the sine routine. For real functions, the input range varies, depending on the value of parm1:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>If parm1 is a 32-bit single floating-point number:</td>
<td>(</td>
</tr>
<tr>
<td>If parm1 is a 64-bit double floating-point number:</td>
<td>(</td>
</tr>
<tr>
<td>If parm1 is an extended floating-point number:</td>
<td>(</td>
</tr>
</tbody>
</table>
For complex functions, the input range differs for the imaginary and real parts of the input.

| For the imaginary part: | \(|\text{Im}(\text{parm1})| < 174.673\) |
|------------------------|----------------------------------------|
| For the real part: single floating-point complex numbers | \(|\text{Re}(\text{parm1})| < (2^{30} \cdot \pi)\) |
| For the real part: double floating-point complex numbers | \(|\text{Re}(\text{parm1})| < (2^{60} \cdot \pi)\) |
| For the real part: extended floating-point complex numbers | \(|\text{Re}(\text{parm1})| < 2^{100}\) |

\(fc\) (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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UT</td>
<td>2</td>
<td>2013</td>
<td>The absolute value of the imaginary part of the parameter was greater than (\text{limit}) in math routine (\text{routine-name}).</td>
</tr>
<tr>
<td>CEE1V1</td>
<td>2</td>
<td>2017</td>
<td>The absolute value of the parameter was greater than or equal to (\text{limit}) in math routine (\text{routine-name}).</td>
</tr>
<tr>
<td>CEE1V3</td>
<td>2</td>
<td>2019</td>
<td>The absolute value of the real part of the parameter was greater than or equal to (\text{limit}) in math routine (\text{routine-name}).</td>
</tr>
</tbody>
</table>

\(result\) (output)

The result of the sine routine. The output range for functions of real variables is: \(result\) is greater than or equal to -1 and is less than or equal to 1 (that is, \(-1 \leq result \leq 1\)).

**CEESxSNH—Hyperbolic sine**

CEESxSNH performs the mathematical function hyperbolic sine by using the equation: \(result = \sinh(\text{parm1})\)

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**

```plaintext
>>>CEESxSNH(--parm1--,--fc--,--result--)<<<
```
CEESxSNH

parm1 (input)
The input to the hyperbolic sine routine. The input range varies.

| Input range for reals: | |parm1| < 175.366 |

For complex functions, the input range differs for the imaginary and real parts of the input. The input range of the imaginary part also differs depending on the precision of parm1.

| For complex functions: real part | |Re(parm1)| < 174.673 |
| For imaginary part: single floating-point complex numbers | |Im(parm1)| < 2^{18} \cdot \Pi |
| For imaginary part: double floating-point complex numbers | |Im(parm1)| < 2^{50} \cdot \Pi |
| For imaginary part: extended floating-point complex numbers: | |Im(parm1)| < 2^{100} |

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1V0</td>
<td>2</td>
<td>2016</td>
<td>The absolute value of the parameter was greater than limit in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
The result of the hyperbolic sine routine. The output range for functions of real variables is:

| result| \leq \Omega |

CEESxSQT—Square root
CEESxSQT returns the square root of parm1 by using the equation:

\[
\text{result} = \sqrt{\text{parm1}}
\]

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
CEESxSQT

128-bit extended floating-point complex number

Syntax

```
>>> CEESxSQT(parm1, fc, result)
```

parm1 (input)
The input to the square root routine. The input range for real number functions is: parm1 is greater than, or equal to, 0 (that 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UQ</td>
<td>2</td>
<td>2010</td>
<td>The parameter was less than limit in math routine routine-name.</td>
</tr>
</tbody>
</table>

result (output)
The result of the square root routine. The output range for functions of real variables is:

\[0 \leq \text{result} \leq \Omega^{\frac{1}{2}}\]

For single-precision routines, \(\Omega = 16^{63}(1 -16^{-6})\)
For double-precision routines, \(\Omega = 16^{62}(1 -16^{-6})\)
For extended-precision routines, \(\Omega = 16^{60}(1 -16^{-6})\)

CEESxTAN—Tangent
CEESxTAN returns the tangent of the parameter by using the equation: \(\text{result} = \tan(parm1)\)

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
- CEESQTN 128-bit extended floating-point number
- CEESTTAN 32-bit single floating-point complex number
- CEESETAN 64-bit double floating-point complex number
- CEESRTAN 128-bit extended floating-point complex number
CEESxTAN

**parm1 (input)**
The input, in radians, to the tangent routine. The input range varies, depending on the value of **parm1**:

| If **parm1** is a single floating-point number: | \[|\text{parm1}| < 2^{18}\times\pi\] |
| If **parm1** is a double floating-point number: | \[|\text{parm1}| < 2^{50}\times\pi\] |
| If **parm1** is an extended floating-point number: | \[<|\text{parm1}| < 2^{100}\] |

Also, for extended and as complex functions, this argument cannot approach odd multiples of:

\[\pi/2\]

**fc (output)**
A 12-byte feedback code, optional in some languages, that indicates the result of this service. The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE1UI</td>
<td>2</td>
<td>2002</td>
<td>The parameter value was too close to one of the singularities (plus or minus (\pi/2), plus or minus (3\pi/2), for the tangent; or plus or minus (\pi), plus or minus (2\pi), for the cotangent) in math routine <strong>routine-name</strong>.</td>
</tr>
<tr>
<td>CEE1V1</td>
<td>2</td>
<td>2017</td>
<td>The absolute value of the parameter was greater than or equal to (\text{limit}) in math routine <strong>routine-name</strong>.</td>
</tr>
<tr>
<td>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine <strong>routine-name</strong>.</td>
</tr>
</tbody>
</table>

**result (output)**
The result of the tangent routine. The output range for functions of real variables is:

\[|\text{result}| \leq \Omega\]

**CEESxTNH—Hyperbolic tangent**
CEESxTNH performs the mathematical function hyperbolic tangent by using the equation: \(\text{result} = \tanh(\text{parm1})\)

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
CEESxTNH

**Syntax**

```
CEESxTNH(parm1, fc, result)
```

**parm1 (input)**
The input to the hyperbolic tangent routine. The input range is not restricted for real functions. For complex functions, `parm1` must not approach odd multiples of:

\[ \pi/2 \cdot i, \text{where } i = \sqrt{-1} \]

**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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td></td>
<td>The service completed successfully.</td>
</tr>
</tbody>
</table>

**result (output)**
The result of the hyperbolic tangent routine. The output range for functions of real variables is:

\[ |result| < 1 \]

**CEESxXPx—Exponentiation**

CEESxXPx performs the mathematical function exponentiation by using the equation:

\[ result = parm1^{parm2} \]

The following routines are provided for the various data types supported:

- **CEESIXPI** 32-bit binary integer raised to a 32-bit binary integer
- **CEESSXPI** 32-bit single floating-point number raised to a 32-bit binary integer
- **CEESDXPI** 64-bit double floating-point number raised to a 32-bit binary integer
- **CEESQXPI** 128-bit extended floating-point number raised to a 32-bit binary integer
- **CEESTXPI** 32-bit single floating-point complex number raised to a 32-bit binary integer
CEESxXPx

CEESEXPI
64-bit double floating-point complex number raised to a 32-bit binary integer

CEESRXPI
128-bit extended floating-point complex number raised to a 32-bit binary integer

CEESSXPS
32-bit single floating-pointing point raised to a 32-bit single floating-point

CEESDXPD
64-bit double floating-point raised to a 64-bit double floating-point

CEESQXPQ
128-bit extended floating-point raised to a 128-bit extended floating-point

CEESTXPT
32-bit single floating-point complex raised to a 32-bit single floating-point complex

CEESEXPE
64-bit double floating-point complex raised to a 64-bit double floating-point complex

CEESRXPR
128-bit extended floating-point complex raised to a 128-bit extended floating-point complex

Syntax

```
->CEESxXPx(---parm1---,---parm2---,---fc---,---result---)
```

parm1 (input)
The input for the base of the exponentiation routine. The input range varies.

<table>
<thead>
<tr>
<th>For functions of real variables:</th>
<th>If parm1 = 0, then parm2 &gt; 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>If parm1 is a 32-bit number and parm1 &lt; 0:</td>
<td></td>
</tr>
<tr>
<td>If parm1 is a 64-bit number and parm1 &lt; 0:</td>
<td></td>
</tr>
<tr>
<td>If parm1 is a 128-bit number and parm1 &lt; 0:</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>Code</td>
<td>Severity</td>
<td>Message Number</td>
<td>Message Text</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CEE1UJ</td>
<td>2</td>
<td>2003</td>
<td>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 <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1UK</td>
<td>2</td>
<td>2004</td>
<td>For an exponentiation operation ((R^I)) where (R) is real and (I) is an integer, (R) was equal to zero and (I) was less than or equal to zero in math routine <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1UL</td>
<td>2</td>
<td>2005</td>
<td>The value of the parameter was outside the valid range <code>range</code> in math routine <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1UM</td>
<td>2</td>
<td>2006</td>
<td>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 <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1UN</td>
<td>2</td>
<td>2007</td>
<td>The exponent exceeded <code>limit</code> in math routine <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1UO</td>
<td>2</td>
<td>2008</td>
<td>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 <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1V4</td>
<td>2</td>
<td>2020</td>
<td>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 <code>limit</code> in math routine <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1V5</td>
<td>2</td>
<td>2021</td>
<td>For an exponentiation operation ((X^Y)), the parameter combination of (Y)\log_2(X) generated a number greater than or equal to <code>limit</code> in math routine <code>routine-name</code>.</td>
</tr>
<tr>
<td>CEE1V8</td>
<td>2</td>
<td>2024</td>
<td>An overflow has occurred in math operation ((X^Y)).</td>
</tr>
<tr>
<td>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine <code>routine-name</code>. The output value from the math routine is undefined.</td>
</tr>
<tr>
<td>CEE1VF</td>
<td>2</td>
<td>2031</td>
<td>The value of the argument was a non-positive whole number in math routine ((X^Y)).</td>
</tr>
</tbody>
</table>

**result (output)**

The result of the exponentiation routine. The output range for functions of real variables is:

\[ |\text{result}| \leq \Omega \]

**Examples of math services**

The following sections contain examples of calls to various math services made from supported languages.
### C/C++ math service examples

Table 29 shows code examples of calling various math services from C/C++.

**Table 29. C/C++ examples**

<table>
<thead>
<tr>
<th>Function called</th>
<th>Code example</th>
</tr>
</thead>
</table>
| Log base 10 and modular arithmetic (CEESDLG1 and CEESIMOD) | ```c
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main (void) {

_FLOAT8 f1,result;
_INT4 int1, int2, intr;
_FEEDBACK fc;
#define SUCCESS "\0\0\0\0"

f1 = 1000.0;
CEESDLG1(&f1,&fc,&result);
if (memcmp(&fc,SUCCESS,4) != 0) {
    printf("CEESDLG1 failed with message number %d\n",fc.tok_msgno);
    exit(2999);
}
printf("%f log base 10 is %f\n",f1,result);

int1 = 39;
int2 = 7;
CEESIMOD(&int1,&int2,&fc,&intr);
if (memcmp(&fc,SUCCESS,4) != 0) {
    printf("CEESIMOD failed with message number %d\n",fc.tok_msgno);
    exit(2999);
}
printf("%d modulo %d is %d\n",int1,int2,intr);
} ``` |

### COBOL math service examples

Table 30 shows code examples of calling various math services from COBOL.

**Table 30. COBOL examples**

<table>
<thead>
<tr>
<th>Function called</th>
<th>Code example</th>
</tr>
</thead>
</table>
| Log base e (CEESSLOG) | ```
77 ARGISRS COMP-1.
77 FBCODE PIC X(12).
77 RESLTRS COMP-1.

CALL "CEESSLOG" USING ARGISRS, FBCODE, RESLTRS.
``` |
| Log base 10 (CEESDLG1) | ```
77 ARGIRL COMP-2.
77 FBCODE PIC X(12).
77 RESLTRL COMP-2.

CALL "CEESDLG1" USING ARGIRL, FBCODE, RESLTRL.
``` |
Table 30. COBOL examples (continued)

<table>
<thead>
<tr>
<th>Function called</th>
<th>Code example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponentiation</td>
<td>77 ARG1IS PIC S9(9) COMP. 77 ARG2IS PIC S9(9) COMP. 77 FBCODE PIC X(12). 77 RESLTIS PIC S9(9) COMP.</td>
</tr>
<tr>
<td></td>
<td>CALL &quot;CEESIXPI&quot; USING ARG1IS, ARG2IS, FBCODE, RESLTIS.</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>77 ARG1RS COMP-1. 77 ARG2IS PIC S9(9) COMP. 77 FBCODE PIC X(12). 77 RESLTRS COMP-1.</td>
</tr>
<tr>
<td></td>
<td>CALL &quot;CEESSXPI&quot; USING ARG1RS, ARG2IS, FBCODE, RESLTRS.</td>
</tr>
<tr>
<td>Arctangent2</td>
<td>77 ARG1RS COMP-1. 77 ARG2RS COMP-1. 77 FBCODE PIC X(12). 77 RESLTRS COMP-1.</td>
</tr>
<tr>
<td></td>
<td>CALL &quot;CEESSAT2&quot; USING ARG1RS, ARG2RS, FBCODE, RESLTRS.</td>
</tr>
</tbody>
</table>

Table 31 shows code examples of calling various math services from PL/I.

Table 31. PL/I examples

<table>
<thead>
<tr>
<th>Function called</th>
<th>Code example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular arithmetic and log base e (CEESIMOD and CEESSLOG)</td>
<td>PLIMATH: PROC OPTIONS(MAIN);</td>
</tr>
<tr>
<td></td>
<td>DCL CEESSLOG ENTRY OPTIONS(ASM) EXTERNAL;</td>
</tr>
<tr>
<td></td>
<td>DCL CEESIMOD ENTRY OPTIONS(ASM) EXTERNAL;</td>
</tr>
<tr>
<td></td>
<td>DCL ARG1 RESULT REAL FLOAT DEC (6);</td>
</tr>
<tr>
<td></td>
<td>DCL ARG1M ARG2M RES2 FLOAT BINARY(21)</td>
</tr>
<tr>
<td></td>
<td>DCL FC CHARACTER (12);</td>
</tr>
<tr>
<td></td>
<td>/* Call log base e routine, which has only one input parameter */</td>
</tr>
<tr>
<td></td>
<td>CALL CEESSLOG (ARG1, FC, RESULT)</td>
</tr>
<tr>
<td></td>
<td>IF ( FC = '0000000000000000000000000000000000'X )</td>
</tr>
<tr>
<td></td>
<td>THEN DO;</td>
</tr>
<tr>
<td></td>
<td>PUT SKIP LIST</td>
</tr>
<tr>
<td></td>
<td>{ 'Error occurred in call to CEESSLOG.' }</td>
</tr>
<tr>
<td></td>
<td>ELSE;</td>
</tr>
<tr>
<td></td>
<td>/* Call modular arithmetic routine, which has two input parameters */</td>
</tr>
<tr>
<td></td>
<td>CALL CEESIMOD (ARG1M, ARG2M, FC, RES2);</td>
</tr>
<tr>
<td></td>
<td>IF ( FC = '0000000000000000000000000000000000'X )</td>
</tr>
<tr>
<td></td>
<td>THEN DO;</td>
</tr>
<tr>
<td></td>
<td>PUT SKIP LIST</td>
</tr>
<tr>
<td></td>
<td>{ 'Error occurred in call to CEESIMOD.' }</td>
</tr>
<tr>
<td></td>
<td>ELSE;</td>
</tr>
<tr>
<td></td>
<td>END;</td>
</tr>
</tbody>
</table>
### Table 31. PL/I examples (continued)

<table>
<thead>
<tr>
<th>Function called</th>
<th>Code example</th>
</tr>
</thead>
</table>
| Double-precision complex tangent (CEESETAN) | TRYETAN: PROCEDURE OPTIONS( MAIN );
    DECLARE FC CHARACTER(12);
    DECLARE PARM1 COMPLEX FLOAT BINARY(53);
    DECLARE RESULT COMPLEX FLOAT BINARY(53);
    DECLARE CEESETAN ENTRY(COMPLEX FLOAT BINARY(53), *, COMPLEX FLOAT BINARY(53)) OPTIONS(ASSEMBLER) EXTERNAL;
    PARM1 = COMPLEX(7,1.1);
    CALL CEESETAN ( PARM1, FC, RESULT);
    IF ( FC ¬= '000000000000000000000000'X) THEN
        PUT SKIP LIST( 'Error in call to CEESETAN.');
    ELSE
        PUT SKIP LIST( 'Result is ' || RESULT);
    END TRYETAN; |
Part 3. Appendixes
Appendix A. IBM-supplied country code defaults

Table 32 contains the currency symbols and default picture strings for the country_code parameters of the COUNTRY run-time option and the national language support callable services. See "COUNTRY" on page 22 and the services listed in Table 17 on page 109 for more information.

**Note:** In the table, some currency symbols are shown as hexadecimal strings. How these are displayed depends on the codeset in use by the terminal device. For example, when using Code Page 01140, X'9F404040' is displayed as the Euro symbol followed by three blanks.

<table>
<thead>
<tr>
<th>Country/Region</th>
<th>Country Code</th>
<th>Decimal Separator</th>
<th>Thousand Separator</th>
<th>Currency Symbol</th>
<th>Int. Currency Symbol</th>
<th>Time Picture String</th>
<th>Date Picture String</th>
<th>Date and Time Picture String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>AF</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>AFN</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Albania</td>
<td>AL</td>
<td>.</td>
<td></td>
<td>Lek</td>
<td>ALL</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Algeria</td>
<td>DZ</td>
<td>.</td>
<td></td>
<td>DZD</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Andorra</td>
<td>AD</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Angola</td>
<td>AO</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>AOA</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Antigua and Barbuda</td>
<td>AG</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>XCD</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Armenia</td>
<td>AM</td>
<td>:</td>
<td>dr:</td>
<td>AMD</td>
<td>dr:</td>
<td>HH:MI:SS</td>
<td>MM/DD/YY</td>
<td>MMMMHHHMMmDD, YYYY HH:MI:SS</td>
</tr>
<tr>
<td>Australia</td>
<td>AU</td>
<td>.</td>
<td></td>
<td>$</td>
<td>AUD</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Austria</td>
<td>AT</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MI:SS,999</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Azerbaijan</td>
<td>AZ</td>
<td>.</td>
<td></td>
<td>AZN</td>
<td>man</td>
<td>HH:MI:SS</td>
<td>MM/DD/YY</td>
<td>ZD MMMMMmnnnnn, YYYY HH:MI:SS</td>
</tr>
<tr>
<td>Bahamas</td>
<td>BS</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>BSD</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Bahrain</td>
<td>BH</td>
<td>.</td>
<td></td>
<td>BHD</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Bangladesh</td>
<td>BD</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>BDT</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Benin</td>
<td>BJ</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>XOF</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Bermuda</td>
<td>BM</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>BMD</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Bolivia</td>
<td>BO</td>
<td>.</td>
<td></td>
<td>BS</td>
<td>BOB</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Bosnia/Herzegovina</td>
<td>BA</td>
<td>.</td>
<td>Dm</td>
<td>BAM</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
<td>.</td>
<td>NCz$</td>
<td>BRL</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Brunei</td>
<td>BN</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>BND</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Bulgaria</td>
<td>BG</td>
<td>.</td>
<td>Lv</td>
<td>BGN</td>
<td>HH:MI:SS</td>
<td>YYYY-RRRZ-DD</td>
<td>YYYY-RRRZ-DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Canada</td>
<td>CA</td>
<td>.</td>
<td>$</td>
<td>CAD</td>
<td>HH:MI:SS,999</td>
<td>YY-MM-DD</td>
<td>YY-MM-DD HH:MI:SS,999</td>
<td></td>
</tr>
<tr>
<td>Cayman Islands</td>
<td>KY</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>KYD</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Chad</td>
<td>TD</td>
<td>.</td>
<td></td>
<td>X'9F404040'</td>
<td>XAF</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
</tbody>
</table>
Table 32. Defaults Currency and Picture Strings Based on COUNTRY Setting (continued)

<table>
<thead>
<tr>
<th>Country/Region</th>
<th>Country Code</th>
<th>Decimal Separator</th>
<th>Thousand Separator</th>
<th>Currency Symbol</th>
<th>Int. Currency Symbol</th>
<th>Time Picture String</th>
<th>Date Picture String</th>
<th>Date and Time Picture String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colombia</td>
<td>CO</td>
<td>,</td>
<td>$</td>
<td>COP</td>
<td>ZH:MM:SS</td>
<td>AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MM:SS AP</td>
</tr>
<tr>
<td>Croatia</td>
<td>HR</td>
<td>,</td>
<td>Din</td>
<td>HRK</td>
<td>HH:MM:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MM:SS</td>
</tr>
<tr>
<td>Dominican</td>
<td>DO</td>
<td>,</td>
<td>$</td>
<td>DOP</td>
<td>ZH:MM:SS</td>
<td>AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MM:SS AP</td>
</tr>
<tr>
<td>Republic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Egypt</td>
<td>EG</td>
<td>,</td>
<td>EGP</td>
<td>HH:MM:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MM:SS</td>
</tr>
<tr>
<td>Finland</td>
<td>FI</td>
<td>,</td>
<td>X'5A404040'</td>
<td>EUR</td>
<td>HH:MM:999</td>
<td>MM/YY</td>
<td>MM/YY</td>
<td>MM/YY HH:MM:999</td>
</tr>
<tr>
<td>Republic of</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td>GR</td>
<td>,</td>
<td>X'FC404040'</td>
<td>GEL</td>
<td>HH:MM:999</td>
<td>MM/YY</td>
<td>MM/YY</td>
<td>MM/YY HH:MM:999</td>
</tr>
<tr>
<td>Guatemala</td>
<td>GT</td>
<td>,</td>
<td>Q</td>
<td>GTQ</td>
<td>HH:MM:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MM:SS</td>
</tr>
<tr>
<td>China (Hong</td>
<td>HK</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HKD</td>
<td>HH:MM:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MM:SS</td>
</tr>
<tr>
<td>Kong S.A.R.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>India</td>
<td>IN</td>
<td>,</td>
<td>X'9F404040'</td>
<td>INR</td>
<td>HH:MM:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MM:SS</td>
</tr>
<tr>
<td>Iran</td>
<td>IR</td>
<td>,</td>
<td>IRR</td>
<td>HH:MM:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MM:SS</td>
</tr>
<tr>
<td>Iraq</td>
<td>IQ</td>
<td>,</td>
<td>IQD</td>
<td>HH:MM:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MM:SS</td>
</tr>
<tr>
<td>Israel</td>
<td>IL</td>
<td>,</td>
<td>NIS</td>
<td>ILS</td>
<td>HH:MM:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MM:SS</td>
</tr>
<tr>
<td>Italy</td>
<td>IT</td>
<td>,</td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MM:999</td>
<td>MM/YY</td>
<td>MM/YY</td>
<td>MM/YY HH:MM:999</td>
</tr>
<tr>
<td>Jordan</td>
<td>JO</td>
<td>,</td>
<td>JOD</td>
<td>HH:MM:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MM:SS</td>
</tr>
<tr>
<td>Latvia</td>
<td>LV</td>
<td>,</td>
<td>Rs</td>
<td>LVL</td>
<td>HH:MM:SS</td>
<td>YYYY/DD.RRRZ</td>
<td>YYYY/DD.RRRZ</td>
<td>YYYY/DD.RRRZ HH:MM:SS</td>
</tr>
<tr>
<td>Country/Region</td>
<td>Country Code</td>
<td>Decimal Separator</td>
<td>Thousand Separator</td>
<td>Currency Symbol</td>
<td>Int. Currency Symbol</td>
<td>Time Picture String</td>
<td>Date Picture String</td>
<td>Date and Time Picture String</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Lesotho</td>
<td>LS</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>ZAR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Liberia</td>
<td>LR</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>LRD</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Libya</td>
<td>LY</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>LYD</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>Liechtenstein</td>
<td>LI</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>CHF</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Lithuania</td>
<td>LT</td>
<td>,</td>
<td>,</td>
<td>Lt</td>
<td>LTL</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>LU</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>China (Macau S.A.R.)</td>
<td>MO</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>MOP</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Macedonia, Former Yugoslav Republic of</td>
<td>MK</td>
<td>,</td>
<td>,</td>
<td>Den</td>
<td>MKD</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Madagascar</td>
<td>MG</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>MGA</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Malawi</td>
<td>MW</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>MWK</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Malaysia</td>
<td>MY</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>MYR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Malta</td>
<td>MT</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Mali</td>
<td>ML</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>XOF</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Mauritania</td>
<td>MR</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>MRO</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Mauritius</td>
<td>MU</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>MUR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Mexico</td>
<td>MX</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>MXN</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MI:SS AP</td>
</tr>
<tr>
<td>Monaco</td>
<td>MC</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Morocco</td>
<td>MA</td>
<td>,</td>
<td>,</td>
<td>MAD</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Mozambique</td>
<td>MZ</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>MZN</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Namibia</td>
<td>NA</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>ZAR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Netherlands</td>
<td>NL</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MI:SS</td>
<td>DD-MM-YYYY</td>
<td>DD-MM-YYYY HH:MI:SS</td>
</tr>
<tr>
<td>Netherlands Antilles</td>
<td>AN</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>ANG</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>New Caledonia</td>
<td>NC</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>XPF</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>New Zealand</td>
<td>NZ</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>NZD</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Nicaragua</td>
<td>NI</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>NIO</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Niger</td>
<td>NE</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>XOFR</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Nigeria</td>
<td>NG</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>NGN</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
<td>,</td>
<td>,</td>
<td>kr</td>
<td>NOK</td>
<td>HH:MI:SS,999</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS,999</td>
</tr>
<tr>
<td>Oman</td>
<td>OM</td>
<td>,</td>
<td>,</td>
<td>OMR</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Pakistan</td>
<td>PK</td>
<td>,</td>
<td>,</td>
<td>PKR</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Panama</td>
<td>PA</td>
<td>,</td>
<td>,</td>
<td>B/</td>
<td>PAB</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MI:SS AP</td>
</tr>
<tr>
<td>Paraguay</td>
<td>PY</td>
<td>,</td>
<td>,</td>
<td>Gs</td>
<td>PYG</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>People's Republic of China</td>
<td>CN</td>
<td>,</td>
<td>,</td>
<td>X'5B404040'</td>
<td>CNY</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>Peru</td>
<td>PE</td>
<td>,</td>
<td>,</td>
<td>l/.</td>
<td>PEN</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Philippines</td>
<td>PH</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>PHP</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Poland</td>
<td>PL</td>
<td>,</td>
<td>,</td>
<td>X'E59A4040'</td>
<td>PLN</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Portugal</td>
<td>PT</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>EUR</td>
<td>HH:MI:SS</td>
<td>DD-MM-YYY</td>
<td>DD-MM-YYY HH:MI:SS</td>
</tr>
<tr>
<td>Puerto Rico</td>
<td>PR</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>USD</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MI:SS AP</td>
</tr>
<tr>
<td>Qatar</td>
<td>QA</td>
<td>,</td>
<td>,</td>
<td>QAR</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Country/Region</td>
<td>Country Code</td>
<td>Decimal Separator</td>
<td>Thousand Separator</td>
<td>Currency Symbol</td>
<td>Int. Currency Symbol</td>
<td>Time Picture String</td>
<td>Date Picture String</td>
<td>Date and Time Picture String</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Taiwan</td>
<td>TW</td>
<td>.</td>
<td>,</td>
<td>$</td>
<td>TWD</td>
<td>HH:MI:SS.999</td>
<td>YY/MM/DD</td>
<td>YY/MM/DD HH:MI:SS.999</td>
</tr>
<tr>
<td>Romania</td>
<td>RO</td>
<td>,</td>
<td>.</td>
<td>Lei</td>
<td>RON</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
</tr>
<tr>
<td>Russia</td>
<td>RU</td>
<td>,</td>
<td>Rub</td>
<td>RUB</td>
<td>HHMJSS</td>
<td>DD mm. YYYY g</td>
<td></td>
<td>DD mm. YYYY.999 HHMJSS</td>
</tr>
<tr>
<td>Saint Lucia</td>
<td>LC</td>
<td>,</td>
<td>X’9F404040’</td>
<td>XCD</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Saudi Arabia</td>
<td>SA</td>
<td>,</td>
<td>SAR</td>
<td>HHMJSS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Senegal</td>
<td>SN</td>
<td>,</td>
<td>X’9F404040’</td>
<td>XOFO</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Seychelles</td>
<td>SC</td>
<td>,</td>
<td>X’9F404040’</td>
<td>SCR</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Sierra Leone</td>
<td>SL</td>
<td>,</td>
<td>X’9F404040’</td>
<td>SLL</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Singapore</td>
<td>SG</td>
<td>,</td>
<td>X’9F404040’</td>
<td>SGD</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Slovakia</td>
<td>SK</td>
<td>,</td>
<td>X’D247A240’</td>
<td>SKK</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Slovenia</td>
<td>SI</td>
<td>,</td>
<td>X’9F404040’</td>
<td>EUR</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Somalia</td>
<td>SO</td>
<td>,</td>
<td>X’9F404040’</td>
<td>SOS</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>South Africa</td>
<td>ZA</td>
<td>,</td>
<td>R</td>
<td>ZAR</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Spain</td>
<td>ES</td>
<td>,</td>
<td>X’9F404040’</td>
<td>EUR</td>
<td>HHMJSS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Sri Lanka</td>
<td>LK</td>
<td>,</td>
<td>X’9F404040’</td>
<td>LKR</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Sudan</td>
<td>SD</td>
<td>,</td>
<td>SDD</td>
<td>HHMJSS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suriname</td>
<td>SR</td>
<td>,</td>
<td>X’9F404040’</td>
<td>SRD</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Swaziland</td>
<td>SZ</td>
<td>,</td>
<td>X’9F404040’</td>
<td>SLL</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Sweden</td>
<td>SE</td>
<td>,</td>
<td>kr</td>
<td>SEK</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Switzerland</td>
<td>CH</td>
<td>,</td>
<td>Fr</td>
<td>CHF</td>
<td>HHMJSS</td>
<td>DD MmMmMmMmMm YYYY</td>
<td>DD. MmMmMmMmMm YYYY HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Syria</td>
<td>SY</td>
<td>,</td>
<td>SYP</td>
<td>HHMJSS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tanzania</td>
<td>TZ</td>
<td>,</td>
<td>X’9F404040’</td>
<td>TZS</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Thailand</td>
<td>TH</td>
<td>,</td>
<td>X’70404040’</td>
<td>THB</td>
<td>HHMJSS</td>
<td>DD/MM/YYYY</td>
<td>DD/MM/YYYY HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Togo</td>
<td>TG</td>
<td>,</td>
<td>X’9F404040’</td>
<td>XOF</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Tunisia</td>
<td>TN</td>
<td>,</td>
<td>TND</td>
<td>HHMJSS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turkey</td>
<td>TR</td>
<td>,</td>
<td>YTL</td>
<td>HHMJSS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uganda</td>
<td>UG</td>
<td>,</td>
<td>X’9F404040’</td>
<td>UGX</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Union of Soviet Socialist Republics</td>
<td>SU</td>
<td>,</td>
<td>Rub</td>
<td>RUB</td>
<td>HHMJSS</td>
<td>DD mm. YYYY g</td>
<td>DD mm. YYYY g HHMJSS</td>
<td></td>
</tr>
<tr>
<td>United Arab Emirates</td>
<td>AE</td>
<td>,</td>
<td>AED</td>
<td>HHMJSS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>United Kingdom</td>
<td>GB</td>
<td>,</td>
<td>GB£</td>
<td>HHMJSS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>United States</td>
<td>US</td>
<td>,</td>
<td>$</td>
<td>USD</td>
<td>HHMJSS</td>
<td>MM/DD/YY</td>
<td>MM/DD/YY HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Uruguay</td>
<td>UY</td>
<td>,</td>
<td>NS</td>
<td>UYU</td>
<td>HHMJSS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Vanuatu</td>
<td>VU</td>
<td>,</td>
<td>X’9F404040’</td>
<td>VUV</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Venezuela</td>
<td>VE</td>
<td>,</td>
<td>BsF</td>
<td>VEB</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Western Samoa</td>
<td>WS</td>
<td>,</td>
<td>X’9F404040’</td>
<td>WST</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Yemen</td>
<td>YE</td>
<td>,</td>
<td>YER</td>
<td>HHMJSS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HHMJSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yugoslavia</td>
<td>YU</td>
<td>,</td>
<td>Din</td>
<td>YUM</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Zaire</td>
<td>ZR</td>
<td>,</td>
<td>X’9F404040’</td>
<td>CDF</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Zambia</td>
<td>ZM</td>
<td>,</td>
<td>X’9F404040’</td>
<td>ZMK</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
<tr>
<td>Zimbabwe</td>
<td>ZW</td>
<td>,</td>
<td>X’9F404040’</td>
<td>ZWD</td>
<td>HHMJSS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HHMJSS</td>
<td></td>
</tr>
</tbody>
</table>
Table 32. Defaults Currency and Picture Strings Based on COUNTRY Setting (continued)

<table>
<thead>
<tr>
<th>Country/Region</th>
<th>Country Code</th>
<th>Decimal Separator</th>
<th>Thousand Separator</th>
<th>Currency Symbol</th>
<th>Int. Currency Symbol</th>
<th>Time Picture String</th>
<th>Date Picture String</th>
<th>Date and Time Picture String</th>
</tr>
</thead>
</table>

Notes:
1. The Czechoslovakia country code CS is obsolete. Use either the Czech Republic country code CZ, or the Slovakia country code SK.
2. Country code DE was used for the former Federal Republic of Germany and now represents the settings for Germany. The DD country code, used in the past for the German Democratic Republic, is obsolete and should be replaced by DE.
3. The SU country code is obsolete. Use the following country codes for the appropriate country: Estonia, EE; Latvia, LV; Lithuania, LT; Russian Federation, RU.
Appendix B. Date and time services tables

This appendix contains information to help you use Language Environment date and time callable services. Included are tables for picture term and national language era usage.

### Table 33. Picture Character Terms Used in Picture Strings for Date and Time Services

<table>
<thead>
<tr>
<th>Picture Terms</th>
<th>Explanations</th>
<th>Valid Values</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>1-digit year</td>
<td>0-9</td>
<td>Y valid for output only.</td>
</tr>
<tr>
<td>YY</td>
<td>2-digit year</td>
<td>00-99</td>
<td>YY assumes range set by CEESCEN.</td>
</tr>
<tr>
<td>YYY</td>
<td>3-digit year</td>
<td>000-999</td>
<td>YY and ZYY valid only if used with &lt;JJJJ&gt;, &lt;CCCC&gt;, or &lt;CCCCCC&gt;.</td>
</tr>
<tr>
<td>ZYY</td>
<td>3-digit year within era</td>
<td>1-999</td>
<td></td>
</tr>
<tr>
<td>YYYY</td>
<td>4-digit year</td>
<td>1582-9999</td>
<td></td>
</tr>
<tr>
<td>&lt;JJJJ&gt;</td>
<td>Japanese era name in DBCS characters</td>
<td>Heisei (X'0E458D45BA0F')</td>
<td>Affects YY field: if &lt;JJJJ&gt; specified, YY means the year within Japanese era, for example, 1988 equals Showa 63. See example in Table 34 on page 522.</td>
</tr>
<tr>
<td>&lt;CCCC&gt; &lt;CCCCCCCC&gt;</td>
<td>Era name in DBCS characters</td>
<td>(X'0E4D8256CE0F')</td>
<td>Affects YY field: if &lt;CCCC&gt; specified, YY means the year within the era. See example in Table 34 on page 522.</td>
</tr>
<tr>
<td>MM ZM</td>
<td>2-digit month 1- or 2-digit month</td>
<td>01-12 1-12</td>
<td>For output, leading zero suppressed. For input, ZM treated as MM.</td>
</tr>
<tr>
<td>RRRR RRRZ</td>
<td>Roman numeral month</td>
<td>Ibbl-Xifb (Left justified)</td>
<td>For input, source string is folded to uppercase. For output, uppercase only. January, Feb, ..., Dec.</td>
</tr>
<tr>
<td>MMM</td>
<td>3-char month, uppercase</td>
<td>JAN-DEC</td>
<td>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 M's, up to 20.</td>
</tr>
<tr>
<td>Mmm</td>
<td>3-char month, mixed case</td>
<td>Jan-Dec</td>
<td></td>
</tr>
<tr>
<td>MMMM...M</td>
<td>3-20 char mo., uppercase</td>
<td>January-December</td>
<td></td>
</tr>
<tr>
<td>MMMMMMMMMMMZ</td>
<td>trailing blanks suppressed</td>
<td>JANUARYybb-DECEMBERybb</td>
<td></td>
</tr>
<tr>
<td>Mmmmmmmmmzz</td>
<td>trailing blanks suppressed</td>
<td>Januaryybb-Decemberybb</td>
<td></td>
</tr>
<tr>
<td>DD ZD 000</td>
<td>2-digit day of month 1- or 2-digit day of year (Julian day)</td>
<td>01-31 1-31 001-366</td>
<td>For output, leading zero is always suppressed. For input, ZD treated as DD.</td>
</tr>
<tr>
<td>HH ZH</td>
<td>2-digit hour 1- or 2-digit hour</td>
<td>00-23 0-23</td>
<td>For output, leading zero suppressed. For input, ZH treated as HH. If AP specified, valid values are 01-12.</td>
</tr>
<tr>
<td>Ml</td>
<td>minute</td>
<td>00-59</td>
<td></td>
</tr>
<tr>
<td>ss</td>
<td>second</td>
<td>00-59</td>
<td></td>
</tr>
<tr>
<td>9 99 999</td>
<td>tenths of a second</td>
<td>0-9 00-99 000-999</td>
<td>No rounding.</td>
</tr>
<tr>
<td>aP aP aP</td>
<td>AM/PM indicator</td>
<td>AM or PM am or pm A.M. or P.M. a.m. or p.m.</td>
<td>AP affects HH/ZH field. For input, source string always folded to uppercase. For output, A generates uppercase and a generates lowercase. Output is padded with blanks (unless Z specified) or truncated to match the number of A's, up to 20.</td>
</tr>
<tr>
<td>W</td>
<td>1-char day-of-week</td>
<td>S, M, T, W, T, F, S</td>
<td>For input, ks are ignored. For output, W generates uppercase and w generates lowercase. Output is padded with blanks (unless Z specified) or truncated to match the number of W's, up to 20.</td>
</tr>
<tr>
<td>WWW</td>
<td>3-char day, uppercase</td>
<td>SUN-SAT</td>
<td></td>
</tr>
<tr>
<td>Www</td>
<td>3-char day, mixed case</td>
<td>Sun-Sat</td>
<td></td>
</tr>
<tr>
<td>WWWWWW...W</td>
<td>3-20 char day, uppercase</td>
<td>SUNDAY-SATURDAY</td>
<td></td>
</tr>
<tr>
<td>WWWWWWWWWWWZ</td>
<td>3-20 char day, mixed case</td>
<td>Sunday-Saturday</td>
<td></td>
</tr>
<tr>
<td>WWWWWWWWWWWZ</td>
<td>trailing blanks suppressed</td>
<td>SUNDAYybb-SATURDAYybb</td>
<td></td>
</tr>
<tr>
<td>All others</td>
<td>Delimiters Constants</td>
<td>X'01'-X'FF' (X'00' reserved for Language Environment use)</td>
<td>For input, treated as delimiters between the month, day, year, hour, minute, second, and fraction of a second. For output, copied exactly as is to the target string. Constant designating year in Russia, Estonia, Latvia, Lithuania, and the Russian Federation. Constant designating time in Sweden.</td>
</tr>
</tbody>
</table>
### Table 33. Picture Character Terms Used in Picture Strings for Date and Time Services (continued)

<table>
<thead>
<tr>
<th>Picture Terms</th>
<th>Explanations</th>
<th>Valid Values</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note: If a Z/z could be interpreted as belonging to the preceding character string and to the following string, then it is always considered part of the following string, even if it would be legal with the preceding string but illegal with the following string. For clarity, you should always use a delimiter to define which string the Z/z belongs with. See Table 34 for an example.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 34. Examples of Picture Strings Recognized by Date and Time Services

<table>
<thead>
<tr>
<th>Picture Strings</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYMMDD</td>
<td>880516</td>
<td>1988-5-16 would also be valid input.</td>
</tr>
<tr>
<td>YYYYMMDD</td>
<td>19880516</td>
<td>Showa is a Japanese Era name. Showa 63 equals 1988.</td>
</tr>
<tr>
<td>&lt;JJJJ&gt; MM/DD/DD</td>
<td>Showa 63.05.16</td>
<td></td>
</tr>
<tr>
<td>&lt;CCCC&gt; YY/MM/DD</td>
<td>Minkow 77.05.16</td>
<td></td>
</tr>
<tr>
<td>MMDDYY</td>
<td>050688</td>
<td>Accepts imbedded blanks 1-digit year format (Y) valid for output only</td>
</tr>
<tr>
<td>MM/DD/YY</td>
<td>05/06/88</td>
<td>Z suppresses zeros/blanks</td>
</tr>
<tr>
<td>ZM/ZD/YY</td>
<td>05/06/88</td>
<td></td>
</tr>
<tr>
<td>MM/DD/YYYY</td>
<td>05/06/1988</td>
<td></td>
</tr>
<tr>
<td>MM/DD/Y</td>
<td>05/06/8</td>
<td></td>
</tr>
<tr>
<td>DD.MM.YY</td>
<td>09.06.88</td>
<td></td>
</tr>
<tr>
<td>DD-RRRR-YY</td>
<td>09-VI-88</td>
<td></td>
</tr>
<tr>
<td>DD MMM YY</td>
<td>09 JUN 88</td>
<td></td>
</tr>
<tr>
<td>DD Mmmmnnnnnnn YY</td>
<td>09 June 88</td>
<td></td>
</tr>
<tr>
<td>ZD Mmmmnnnnnnz YY</td>
<td>9 June 88</td>
<td></td>
</tr>
<tr>
<td>Mmmmnnnnnnn ZD, YYYY</td>
<td>June 9, 1988</td>
<td></td>
</tr>
<tr>
<td>YYDDD</td>
<td>88.137</td>
<td>Julian date</td>
</tr>
<tr>
<td>YYDDD</td>
<td>88137</td>
<td></td>
</tr>
<tr>
<td>YYYY/DDD</td>
<td>1988/137</td>
<td></td>
</tr>
<tr>
<td>YYYYMMDDHHMISS</td>
<td>880516204229</td>
<td></td>
</tr>
<tr>
<td>YYYYMMDDHHMISS</td>
<td>19880516204229</td>
<td></td>
</tr>
<tr>
<td>WWW, ZM/ZD/YY HH:MI AP</td>
<td>MON, 5/16/88 08:42 PM</td>
<td></td>
</tr>
<tr>
<td>WWWWWWWWZ DD Mmm YYYY ZZ:MI AP</td>
<td>Monday, 16 May 1988, 8:42 PM</td>
<td></td>
</tr>
</tbody>
</table>

### Table 35. Japanese Eras Used by Date/Time Services When <JJJJ> Specified

<table>
<thead>
<tr>
<th>First Date of Japanese Era</th>
<th>Era Name</th>
<th>Era Name in IBM Japanese DBCS Code</th>
<th>Valid Year Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1868-09-08</td>
<td>Meiji</td>
<td>X’0E45A645840F’</td>
<td>01-45</td>
</tr>
<tr>
<td>1912-07-30</td>
<td>Taisho</td>
<td>X’0E45B45770F’</td>
<td>01-15</td>
</tr>
<tr>
<td>1926-12-25</td>
<td>Showa</td>
<td>X’0E45B3457A0F’</td>
<td>01-64</td>
</tr>
<tr>
<td>1989-01-08</td>
<td>Heisei</td>
<td>X’0E458D458A0F’</td>
<td>01-999 (01 = 1989)</td>
</tr>
</tbody>
</table>
Appendix C. Controlling storage allocation

To generate a report of the storage a routine (or more specifically, an enclave) used during its run, specify the RPTSTG(ON) 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.

Neither the storage report nor the corresponding run-time options include the storage that Language Environment acquires during early initialization, before run-time options processing, and before the start of space management monitoring.

Storage statistics

Attention: This section does not apply to AMODE 64 applications. For information on AMODE 64 statistics, see "Storage statistics for AMODE 64 applications."

The following run-time options control storage allocation:

- ANYHEAP
- BELOWHEAP
- HEAP
- HEAPPOLS
- LIBSTACK
- STORAGE
- STACK
- THREADHEAP
- THREADSTACK

Ensure that these options are tuned appropriately to avoid performance problems. Tuning tips are provided in the z/OS Language Environment Programming Guide. For an example and complete description of the storage report, see z/OS Language Environment Debugging Guide.

Storage statistics for AMODE 64 applications

The following run-time options control storage allocation:

- HEAP64
- HEAPPOLS
- HEAPPOLS64
- IOHEAP64
- LIBHEAP64
- STACK64
- THREADSTACK64

Ensure that these options are tuned appropriately to avoid performance problems. Tuning tips are provided in the z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode. For an example and complete description of the storage report, see z/OS Language Environment Debugging Guide.
Appendix D. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you may view the information through the z/OS Internet Library website or the z/OS Information Center. If you continue to experience problems, send an email to mhvrcfs@us.ibm.com or write to:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer and z/OS ISPF User’s Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer or Library Server versions of z/OS books in the Internet library at:

http://www.ibm.com/systems/z/os/zos/bkserv/

One exception is command syntax that is published in railroad track format, which is accessible using screen readers with the Information Center, as described in "Dotted decimal syntax diagrams."

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always
present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 \* FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* \* FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only
one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:
1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.
Notices

This information was developed for products and services offered in the U.S.A.

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.
Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Programming Interface information

This book documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Language Environment in z/OS.

Trademarks

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

Adobe, Acrobat, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

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

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

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

This section lists the books in the Language Environment library and other publications that may be helpful when using Language Environment.

Language Products Publications

**z/OS Language Environment**
- **z/OS Language Environment Concepts Guide**, SA22-7567
- **z/OS Language Environment Programming Guide**, SA22-7561
- **z/OS Language Environment Programming Reference**, SA22-7562
- **z/OS Language Environment Customization**, SA22-7564
- **z/OS Language Environment Debugging Guide**, GA22-7560
- **z/OS Language Environment Run-Time Application Migration Guide**, GA22-7565
- **z/OS Language Environment Writing Interlanguage Communication Applications**, SA22-7563
- **z/OS Language Environment Run-Time Messages**, SA22-7566
- **z/OS Language Environment Vendor Interfaces**, SA22-7568
- **z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode**, SA22-7569

**z/OS XL C/C++**
- **z/OS XL C/C++ Language Reference**, SC09-4815
- **z/OS XL C/C++ Compiler and Run-Time Migration Guide for the Application Programmer**, GC09-4913
- **z/OS XL C/C++ Programming Guide**, SC09-4765
- **z/OS XL C/C++ User’s Guide**, SC09-4767
- **z/OS XL C/C++ Run-Time Library Reference**, SA22-7821
- **z/OS XL C/C++ Messages**, GC09-4819
- **Standard C++ Library Reference**, SC09-4949

**z/OS Metal C Runtime Library**
- **z/OS Metal C Programming Guide and Reference**, SA23-2225

**Enterprise COBOL for z/OS, V4R2**
- **Enterprise COBOL for z/OS, V4R2, Licensed Program Specifications**, GI11-7871
- **Enterprise COBOL for z/OS, V4R2, Customization**, SC23-8526
- **Enterprise COBOL for z/OS, V4R2, Language Reference**, SC23-8528
- **Enterprise COBOL for z/OS, V4R2, Programming Guide**, SC23-8529
- **Enterprise COBOL for z/OS, V4R2, Compiler and Runtime Migration Guide**, GC23-8527

**Enterprise COBOL for z/OS, V3R4**
- **Enterprise COBOL for z/OS, V3R4, Licensed Program Specifications**, GC27-1411
- **Enterprise COBOL for z/OS, V3R4, Customization**, GC27-1410
- **Enterprise COBOL for z/OS, V3R4, Language Reference**, SC27-1408
- **Enterprise COBOL for z/OS, V3R4, Programming Guide**, SC27-1412
- **Enterprise COBOL for z/OS, V3R4, Compiler and Runtime Migration Guide**, GC27-1409

**COBOL for OS/390 & VM**
- **COBOL for OS/390 & VM Licensed Program Specifications**, GC26-9044
- **COBOL for OS/390 & VM Customization under OS/390**, GC26-9045
Related Publications

CICS
- CICS Transaction Server for z/OS Installation Guide SC34-6224
- CICS Operations and Utilities Guide SC34-6229
- CICS Problem Determination Guide SC34-6239
- CICS Resource Definition Guide SC34-6228
- CICS Application Programming Guide SC34-6231
- CICS Application Programming Reference SC34-6232
- CICS System Definition Guide SC34-6226

DB2
- Database 2 Application Programming and SQL Guide, SC26-4377

DFSMS/MVS
- z/OS MVS Program Management: User’s Guide and Reference SA22-7643
- z/OS DFSMS DFM Guide and Reference SC26-7395

IPCS
- z/OS MVS IPCS User’s Guide SA22-7596
- z/OS MVS IPCS Commands SA22-7594
- z/OS MVS IPCS Customization SA22-7595
DFSORT
- z/OS DFSORT Application Programming Guide SC26-7523

IMS/ESA
- IMS Version 8: Application Programming: Database Manager SC27-1286
- IMS Version 8: Application Programming: Transaction Manager SC27-1289
- IMS Version 8: Application Programming: EXEC DLI Commands for CICS and IMS Version 8 SC27-1288

z/OS
- z/OS Introduction and Release Guide GA22-7502
- z/OS Program Directory GI10-0670
- z/OS Planning for Installation GA22-7504
- z/OS Information Roadmap SA22-7500
- z/OS Hot Topics Newsletter GA22-7501
- z/OS Licensed Program Specifications GA22-7503

z/OS ISPF
- z/OS ISPF Dialog Tag Language Guide and Reference SC34-4824
- z/OS ISPF Planning and Customizing GC34-4814
- z/OS ISPF Dialog Developer’s Guide and Reference SC34-4821

z/OS UNIX System Services
- z/OS UNIX System Services User’s Guide SA22-7801
- z/OS UNIX System Services Command Reference SA22-7802
- z/OS UNIX System Services Programming: Assembler Callable Services Reference SA22-7803
- z/OS UNIX System Services Planning GA22-7800

z/OS TSO/E
- z/OS TSO/E Customization SA22-7783
- z/OS TSO/E Programming Services SA22-7789
- z/OS TSO/E System Programming Command Reference SA22-7793
### Index

#### Special characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ (slash)</td>
<td>NOEXECOPS alters behavior of specifying in parameters of TEST run-time option.</td>
</tr>
<tr>
<td>* (asterisk)</td>
<td>84, 86</td>
</tr>
<tr>
<td>argc parameter for C</td>
<td>ARGPARSE run-time option's effect on 16</td>
</tr>
<tr>
<td>argv parameter for C</td>
<td>ARGPARSE run-time option's effect on 16</td>
</tr>
<tr>
<td>calloc() function</td>
<td>36</td>
</tr>
<tr>
<td>fprintf function</td>
<td>147</td>
</tr>
<tr>
<td>setlocale() function</td>
<td>CEE3CTY callable service and 134</td>
</tr>
<tr>
<td>COUNTRY run-time option and 23</td>
<td></td>
</tr>
<tr>
<td>= (equal sign)</td>
<td>in ENVAR run-time option 29</td>
</tr>
<tr>
<td>' (straight single quote)</td>
<td>initializing storage with 76, 78 specifying in parameters of TEST run-time option 85, 86</td>
</tr>
<tr>
<td>&quot; (straight double quote)</td>
<td>initializing storage with 76, 78 specifying in parameters of TEST run-time option 85, 86</td>
</tr>
</tbody>
</table>

#### Numerics

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>type_of_move 392, 394</td>
</tr>
<tr>
<td>1</td>
<td>type_of_move 392, 394</td>
</tr>
<tr>
<td>16M line</td>
<td>ALL31 run-time option and 13 allocating library heap storage not restricted to below 14, 15</td>
</tr>
<tr>
<td>ANYHEAP run-time option and 14, 15</td>
<td></td>
</tr>
<tr>
<td>HEAP run-time option and 37 storage received from CEEGTST and 331</td>
<td></td>
</tr>
<tr>
<td>2-digit years</td>
<td>querying within 100-year range (CEEQCEN) 414 examples of 415 setting within 100-year range (CEESCEN) 428 examples of 430</td>
</tr>
<tr>
<td>24-bit addressing</td>
<td>ANYHEAP run-time option and 14</td>
</tr>
</tbody>
</table>

#### A

- abbreviating run-time options 9
- abcode parameter 9, 10, 121, 124
- abend codes
  - abend 4088, reason code 1004 77
  - abend 4091, reason code 21 25
  - dumping 121, 125
- additional heap (continued) reducing the amount of storage you need to run an application 298
- address parameter
  - CEECZST callable service and 232
  - CEEFRST callable service and 299
  - CEEGTST callable service and 332
- addressing exception 209 aggregate
- aggregate dumping arrays and structures 144
- AIXBLD run-time option
  - syntax 11
- alignment
  - PL/I structures and Language Environment storage callable services 333
- ALL31 run-time option
  - ALL31(ON) default under CICS 13
  - syntax 12
- allocating storage
  - additional heap, setting size of (CEECHRHP) 227
  - anywhere heap, setting size of (ANYHEAP run-time option) 14, 15
  - below heap, setting size of (BELOWHEAP run-time option) 17, 18
  - getting heap storage (CEEQTST) 332, 334, 335
  - initial heap, setting size of (HEAP run-time option) 36, 48
  - initializing when allocating (STORAGE run-time option) 75, 78
  - library stack storage, setting size of (LIBSTACK run-time option) 51
  - stack storage, setting size of (STACK run-time option) 71, 74, 75, 91
- alternate indices, building with AIXBLD
- NOAIXBLD run-time option 12
- AMODE
  - ALL31 run-time option and 12, 13 callable services and 112
  - heap storage 38
  - STACK run-time option and 13 under CICS 333
- AMS (access method service) 11
- ANYHEAP run-time option
  - syntax 14
- anywhere heap
  - allocating storage from 14, 15
  - ANYHEAP run-time option and 14, 15
  - controlling whether storage is freed from 14
- ANYWHERE suboption
  - ANYHEAP run-time option and 14
  - HEAP run-time option and 37

© Copyright IBM Corp. 1991, 2011
application writer interface (AWI) 111
arcosine math service (CEEExACS) 479
arcosine math service (CEEExASN) 479
arctangent math service (CEEExATN) 481
arctangent math service (CEEExAT2) 482
AREA storage for PL/1 333
ARGPARSE run-time option
  CICS ignores this option 16
  syntax 15
argument
  dumping 144
  format of in invoked routine 61, 62
  list format
  effect of EXECOPS run-time option on 32
arithmetic
  calculation on dates
  convert date to COBOL Lilian format (CEECCBLDY) 218, 220
  convert date to Lilian format (CEEEDAYS) 250, 252
  convert timestamp to number of seconds (CEESECS) 441, 444
  get current Greenwich Mean Time (CEEGMT) 309, 310
examples using 476, 509, 512
  parameter types 475
  services Quick Reference Tables 108, 109
  syntax variations of 108, 109
  syntax 15
array
  assembly language routines
calling math services from 475
  invoking callable services
  from 112
asterisk (*) 84, 86
AT OCCURRENCE debug command for
  Debug Tool 84
AT TERMINATION debug command for
  Debug Tool 84
attention processing
  handling with a debug tool (TEST run-time option) 84
  handling with INTERRUPT run-time option 47
AUTOTASK run-time option
  syntax 16

B

BASEx storage for PL/1 36
below heap
  allocating storage from 17, 18
  BELOWHEAP run-time option and 17, 18
  controlling whether storage is freed 17
BELOW suboption
  ANYHEAP run-time option and 14
  BELOWHEAP run-time option
  syntax 17
  BIF (built-in functions), in
  CEE3DMP 145
  CEE3DMP 145
  CEEEMCR—move resume cursor 395
  CEEMSG—get, format, and dispatch a message 404
  CEEENCOD—construct a condition token 410
  CEEQCN—query current window
    415
  CEERAN0—calculate uniform random numbers 426
  CEESECN—set century
    window 430
  CEESDLG1—logarithm base 10 510
  CEESECN—set century window 430
  CEESDLG1—logarithm base 10 510
  CEESECI—convert seconds to integers 438
  CEESECS—convert timestamp to seconds 445
  CEESSL—signal a condition 159, 456
  CEEISMOD—modular arithmetic 510
  CEEIST—invoke debug tool 467
  fc parm, omitting 114
  hardware interrupts that cannot be enabled 208
  CEEQCN—query current window
    415
  CEERAN0—calculate uniform random numbers 426
  CEESECN—set century
    window 430
  CEESDLG1—logarithm base 10 510
  CEESECI—convert seconds to integers 438
  CEESECS—convert timestamp to seconds 445
  CEESSL—signal a condition 159, 456
  CEEISMOD—modular arithmetic 510
  CEEIST—invoke debug tool 467
  fc parm, omitting 114
  hardware interrupts that cannot be enabled 208
C (continued)

HEAP considerations 38
invoking callable services from 114
mapping SPIE and STAE to TRAP 94
MSGFILE run-time option and 54
 queryInterface function 54
PLIST setting under IMS differs
depending on version of C
compiler 61
run-time options specific to Language
Environment
ARGPARSE 15
ENV 28, 29
EXECOPS 32
PLIST 61
REDIR 66
run-time options, specifying on the
command line 32
stderr
MSGFILE run-time option and 54
REDIR run-time option and 66
redirecting output from 54
variables, where stored 36
call return point 391
callable services
AMODE switching across calls to 13
CEE3ABD—terminate enclave with an
abend 121, 124, 169
CEE3CIB—return pointer to condition
information block 124, 128
CEE3CTY—set default country 132
CEE3DMP—generate dump 142
CEE3GRC—get the enclave return
code 149
CEE3GRN—get name of routine that
incurred condition 158
CEE3GRO—get offset of
condition 164
CEE3LING—set national
language 174
CEE3MCS—get default currency
symbol 181
CEE3MDS—get default decimal
separator 189
CEE3MTS—get default thousands
separator 192
CEE3PRM—query parameter
string 195
CEE3RPH—set report heading 203
CEE3SPM—query and modify
Language Environment hardware
condition enablement 206
CEE3SRC—set the enclave return
code 212
CEE3SRP—set resume point 213
CEE3USR—set or query user area
fields 214
CEE3BLDLY—convert date to COBOL
Lilian format 218
CEE3CM1—store and load message
insert data 222
CEE3CRHP—create new additional
heap 227
CEE3CST—reallocate (change size of)
storage 232
CEE3DATE—convert Lilian date to
character format 237
CEE3DAYM—convert seconds to
calendar timestamp 243
CEE3DAYS—convert date to Lilian
format 250
CEE3DCOD—decompose a condition
token 256
CEE3DHP—discard heap 268
CEE3DYWK—calculate day of week
from Lilian date 272
CEE3FMA—get default date
format 284
CEE3FMDT—get default date and time
format 287
CEE3MON—format monetary
string 290
CEE3TMT—get default time
format 295
CEE3TM—get current Greenwich
Time 309
CEE3GMT—get current Greenwich
Mean Time 309
CEE3GMTO—get offset from
Greenwich Mean Time to local
time 313
CEE3GPID—retrieve Language
Environment version and platform
ID 316
CEE3QDT—retrieve
q_data_token 321
CEE3GST—get heap storage 331
CEE3DLR—register user condition
handler 336
CEE3DLU—unregister user condition
handler 346
CEE3ISEC—convert integers to
seconds 351
CEE3ITOK—return initial condition
token 357
CEE3CNV—query locale numeric
conventions 363
CEE3COT—get current local
time 370
CEE3GET—get a message 374
CEE3MOUT—dispatch a message 380
CEE3MRCE—move resume cursor
explicit 384
CEE3MRCR—move resume cursor
relative to handle cursor 391
CEE3MSG—get, format, and dispatch a
message 402
CEE3NCOD—construct a condition
token 407
CEE3QCN—query the century
window 414
CEE3QDTC—query locale date and
time conventions 417
CEE3QRYL—query active locale
environment 423
CEE3RN0—calculate uniform random
numbers 425
CEE3SCN—set the century
window 428
CEE3COL—compare string collation
weight 432
CEE3EIC—convert seconds to
integers 436
CEE3EC—convert timestamp to
number of seconds 441
CEE3ETL—set the locale operating
environment 448
CEE3EGL—signal a condition 453
CEE3CNR—bit clear 471
CEE3SET—bit set 471
CEE3ST—bit test 473
CEE3TX—transform string characters
into collation weights 459
CEE3ABS—absolute value 477
CEE3ACS—arccosine 478
CEE3ASN—arc sine 479
CEE3A2—arctangent of two
arguments 482
CEE3ATH—hyperbolic
arctangent 480
CEE3ATN—arctangent 481
CEE3CJG—conjugate complex 483
CEE3COS—cosine 484
CEE3CSH—hyperbolic cosine 485
CEE3CTN—cotangent 486
CEE3DIM—positive difference 487
CEE3DVD—division 488
CEE3ERC—error function 489
CEE3ERF—error function
complex 490
CEE3EXP—exponent (base e) 490
CEE3GMA—gamma function 492
CEE3IM—imaginary part of
complex 493
CEE3INT—truncation 493
CEE3LG1—logarithm base 10 495
CEE3LG2—logarithm base 2 496
CEE3LGM—log gamma
function 494
CEE3LOG—logarithm base e 497
CEE3M—floating complex
multiply 498
CEE3MOD—modular
arithmetic 498
CEE3NIN—nearest integer 499
CEE3NWN—nearest whole
number 500
CEE3SGN—transfer of sign 501
CEE3SIN—sine 502
CEE3SINH—hyperbolic sine 503
CEE3SQRT—square root 504
CEE3TAN—tangent 505
CEE3TANH—hyperbolic tangent 506
CEE3XP—exponential (e^x) 507
CEE3DLI—invoke IMS 463
CEE3TEST—invoke debug tool 466
CEE3TEST—invoke debug tool
in C 112, 114
CEE3T—invoke IMS in COBOL 115, 116
in PL/I 116, 117
Quick Reference Tables 105, 108
case 1 condition token 409
case 2 condition token 409
cause condition code 409
Index 539
CEE3MCS—get default currency symbol
(continued)
default if invalid country_code
specified 182, 186
examples using 183, 185
syntax 181
table of default values for specified
country_code 515
CEE3MDS—get default decimal separator
CEE3CTY callable service and 189
COUNTRY run-time option and 189
default if invalid country_code
specified 190
defaults of country_code
parameter 515
examples using 191, 192
syntax 189
CEE3MTS—get default thousands
separator
CEE3CTY callable service and 192
COUNTRY run-time option and 192
default if invalid country_code
specified 193
examples using 193
syntax 192
table of default values for specified
country_code 515
CEE3PRM—query parameter string 414
examples using 196, 198
syntax 195
CEE3PRH—set report heading
examples using 203
syntax 203
CEE3SPM—query and modify Language
Environment hardware condition
enablement
examples using 209, 210
syntax 206
CEE3SRC—set the enclave return code
examples using 213
syntax 212
CEE3SRP—set resume point
examples using 214
syntax 213
CEE3USR—set or query user area fields
examples using 215, 218
syntax 214
CEE3BLDX EXEC
use caution when creating new facility
ID for 408
CEE3XITA assembler user exit
percolating certain abends with
TRAP(ON) 94
CEE3BLDV—convert date to COBOL
Lilian format 218
examples using 219
syntax 218
CEE3CMI—store and load message insert
data 222
examples using 225, 227
syntax 222
CEE3CRH—create new additional heap
examples using 230, 232
HEAP run-time option and 332
syntax 227
CEE3ZST—reallocating (change size of) storage
CEE3TST callable service and 233
examples using
example with CEE3TST and
CEE3FRST 234
examples with CEE3HDR, CEE3TST, and CEE3RJR 234, 236
syntax 233
CEE3DATE—convert Lilian date to character format
CEE3DAYS callable service and 237
CEE3MST callable service and 238
COUNTRY run-time option and 238
examples using 239, 243
syntax 237
table of sample output 243
CEE3DATM—convert seconds to character
timestamp
CEE3MTT callable service and 244
CEE3FMT callable service and 436
CEE3ESCS callable service and 243
COUNTRY run-time option and 244
examples using 245, 249
syntax 243
table of sample output 249
CEE3DAYS—convert date to Lilian format
CEE3DATE callable service and 250
CEE3ECN callable service and 218, 250
syntax 250
CEE3ECOD—decompose a condition
token
examples using
C example showing alternative to
using 456
examples with CEE3TST 259, 261
syntax 256
CEE3DOPT
CEE3OPTS run-time option must be
specified in CEE3DOPT or
CEE3UOPT 18
COUNTRY run-time option
consideration 23
country_code and 22
ENVAR run-time option and 30
national language codes and 56, 174
specifying nonexistent national
language code in 56
specifying UPSI run-time option
in 96
CEE3DSHP—discard heap
can overrule HEAP run-time
option 268
CEE3CHP callable service and 268
examples with CEE3CHP 269, 271
HEAP run-time option and 268
syntax 268
CEE3DUMP default dump file
CBLQDA run-time option and 19
CEE3MST callable service and 142
CEE3DYWK—calculate day of week from
Lilian date
examples using 273, 277
syntax 272
CEE3EMAW file, description of 113
**Index**

CEEEDCCT file, description of 113
CEEFMADA—get default date format 284
CEECSTY callable service 284
COUNTRY run-time option and default if invalid country_code specified 284
defaults of country_code parameter 515
examples using 285, 287
syntax 284
CEEFMDT—get default date and time format
CEECSTY callable service and 287
COUNTRY run-time option and default if invalid country_code specified 287
defaults of country_code parameter 515
examples using 288, 289
syntax 287
CEEFMON—format monetary string about 290
examples using 293
CEEFMFTM—get default time format
CEECSTY callable service and 295
COUNTRY run-time option and default if invalid country_code specified 295
defaults of country_code parameter 515
examples using 296, 298
syntax 295
CEEFRST—free heap storage
CEECRHP callable service and 299
CEEGST callable service and 298
examples with CEEGST 300, 301
HEAP run-time option and 299
syntax 298
CEEFTDS—format date and time into character string
syntax 303
CEEGETM—get current Greenwich Mean Time 470
CEEDATE callable service and 310
CEEDATM callable service and 310
CEEGMTO callable service and 309
examples using 310, 313
syntax 309
CEEGETL—get offset from Greenwich Mean Time to local time
CEEDATM callable service and 314
examples using 314, 315
syntax 313
CEEGBPID—retrieve the Language Environment version and platform ID examples using 317, 321
syntax 316
CEEQDT—retrieve q_data_token
CEESGL callable service and 321
elements using CEEQDT by itself 324
examples with CEEDHDLR and CEESGL 322, 324
instance specific information (ISI) and 321
syntax 321
CEEGETST—get heap storage
CEECRHP callable service and 332
CEEDHHP callable service and 331
CEEFRST callable service and 331
CEESGL callable service and 331
examples using
examples with CECZST and CEEFRST 234
examples with CEEFRST 300, 334, 335
HEAP run-time option and 332
syntax 331
using STORAGE run-time option to initialize storage received from 331
CEEHDLR—register user condition handler
CEEHDLR callable service and 347
condition handling example 337
examples using
CEEHDLR by itself 338, 345
examples with CEESRC and CEEGRC 159, 164
syntax 336
CEEHDLU—unregister user condition handler
CEEHDLR callable service and 347
examples with CEEHDLR 348, 350
syntax 346
CEEBMCI file, description of 113
CEEBMCIT file, description of 113
CEEGZCI file, description of 113
CEEGZCT file, description of 113
CEEISEC—convert integers to seconds
CEESCI callable service and 352
examples using 354, 356
syntax 351
CEEITOK—return initial condition token examples using 359, 363
syntax 357
CEECNV—query locale numeric conventions
syntax 363
CEEDAYS callable service and 428
syntax 428
CEESECL—convert string collation weight
syntax 432
CEESECI—convert seconds to integers
examples using 438, 441
relationship to CEESECI callable service 436
syntax 436
CEESECS—convert timestamp to number of seconds
CEEDATM callable service and 441
CEESESEC callable service and 351
CEESSCN callable service and 428
COUNTRY run-time option and 443
syntax 441
CEESETL—set locale operating environment
syntax 448
CEESGL—signal a condition
CEEGQDT callable service and 321
examples using 456, 459
HLL-specific condition handlers and 454
q_data_token and 321, 454
setting of ERRCOUNT run-time option can cause abend 454
syntax 453
TRAP run-time option does not affect 93
using to create an ISI 453, 454
CEEESTXF—transform string into collation weights
syntax 459
CEEFDATA DD statement restrictions 7 using 7
CEEQCN—query the century window
CEESCN callable service and 414
examples using 415, 417
syntax 414
CEEQDC—query locale, date, and time conventions
syntax 417
CEEQRYL—query active locale environment
syntax 423
CEERAN0—calculate uniform random numbers
examples using 426, 428
syntax 425
CEESCN—set the century window
CEEDAYS callable service and 428
CEESECS callable service and 428
examples using 430, 432
syntax 428
CEESCOL—compare string collation
weight
syntax 432
CEESRC—convert seconds to integers
examples using 438, 441
relationship to CEESRC callable service 436
syntax 436
CEESCS—convert timestamp to number of seconds
CEEDATM callable service and 441
CEESESEC callable service and 351
CEESSCN callable service and 428
COUNTRY run-time option and 443
syntax 441
CEESTL—set locale operating environment
syntax 448
CEESGL—signal a condition
CEEGQDT callable service and 321
examples using 456, 459
HLL-specific condition handlers and 454
q_data_token and 321, 454
setting of ERRCOUNT run-time option can cause abend 454
syntax 453
TRAP run-time option does not affect 93
using to create an ISI 453, 454
CEEESTXF—transform string into collation weights
syntax 459
CEETDLI interface to IMS
syntax 463
CEETEST—invoke debug tool
elements using 467, 469
NOTEST run-time option and 86
syntax 466
CEEUOPT
CBLOPTS run-time option must be specified in CEEUOPT or
CEEDOPT 18
COUNTRY run-time option
considerations 23
country_code and 22
national language codes and 56, 174
specifying nonexistent national
language code in 23
specifying UPSI run-time option
in 24, 96
century window
CEEDAYS callable service and 218, 250
CEEQCEN callable service and 414
examples of 415, 417
CEESCEN callable service and 428
examples of 415, 417, 430, 432
CEESSC callable service and 441
CEES transient data queue
CEEMOUT callable service and 381
TERMTHDACT run-time option
and 177
char_parm_string parameter 195
character timestamp
converting Lilian seconds to
(CEEDATM) 243
examples of 245, 249
converting to COBOL Lilian seconds
(CEECBLDY) 218
examples of 221
converting to Lilian seconds
(CEESECS)
examples of 441
CHARn Language Environment data
type and HLL equivalents 117
CHECK run-time option
syntax 22
using while debugging 22
checking errors, flagging with CHECK
run-time option 22
ChuHwaMinKow era 523
CIB (Condition Information Block) 124, 128
CICS
callable service behavior under
abend codes in CEE3ABD callable
service 121, 124
storage considerations 333
CBLPSHPOP run-time option and 18
CEEDMPS ENCLAVE
considerations 146
CESE transient data queue and 54
PLIST setting 61
string 293
class code condition 409
clean-up parameter 121, 124
CMS
CEEDMPS special considerations 146
CMS (continued)
CMSTOR OBTAIN/RELEASE
commands 68
OSRUN command 61
PLIST run-time option settings to
specify when running under 61
specifying a C application is running under
28, 32
CMSCALL
CEEDMPS ENCLAVE
considerations 146
CMSTOR OBTAIN/RELEASE 68
COBOL
batch debugging features 23
callable services, invoking from 115, 116
declares for Language Environment
data types 117
examples
CEEDMPS—terminate enclave with
abend 123, 126, 171
CEEDCTY—set default
country 135
CEEDMPS—generate dump 147
CEEDGRC—get enclave return
code 150, 157
CEEDGRN—get name of routine
that incurred condition 160
CEEDLING—set national
language 177
CEEDMCS—get default
currency 183
CEEDMDS—get default decimal
separator 191
CEEDMSE—get default thousands
separator 193
CEEDPRM—query parameter
string 196
CEEDPHE—set report
heading 204
CEEDPSM—query and modify
Language Environment hardware
condition enablement 209
CEEDSRC—set enclave return
code 150, 157
CEEDUSR—set or query user area
fields 215
CEEDFCMI—store and load message
insert data 225
CEEDFRHP—create new additional
heap 230
CEEDCST—reallocates storage 234
CEEDATE—convert Lilian date to
character format 240
CEEDATM—convert seconds to
counter time 246
CEEDAYS—convert date to Lilian
counter time 247
CEEDCOD—decompose a
condition token 259
CEEDFON—format monetary
string 293
CEEDFTDS—format date and time
into character string 306
CEEDPID—retrieve Language
Environment version/platform
ID 318
COBOL (continued)
examples (continued)
CEEDQDFT—retrieve
q_data_token 324
CEEDGST—get heap storage 234, 334
CEEDTDR—register user-written
counter handler 150, 157, 338
CEEDTDLU—unregister
user-written condition
handler 348
CEEDSEC—convert integers to
seconds 354
CEEDTOK—return initial condition
token 359
CEEDCNV—query locale numeric
conventions 367
CEEDTST—get current local date
or time 372
CEEDMGET—get a message 378
CEEDMOUT—dispatch a
message 382
CEEDMCCR—move resume
cursor 397
CEEDMSG—get, format, and
dispatch a message 404
CEEDCOE—construct a condition
token 410
CEEDCNV—query century
window 415
CEEDQDTC—return locale date and
time 419
CEEDQRYL—query active locale
environment 424
CEEDMSEN—set century
window 430
CEEDCML—compare string
collation weight 432
CEEDEC—convert seconds to
integers 438
CEEDEC—convert timestamp to
seconds 445
CEEDSTL—set locale operating
environment 434, 450
CEEDSG—signal a condition 456
CEEDSLOG—calculate logarithm
base e 510
CEEDSTX—transform string
characters into collation
weights 461
CEEDTST—invoke a debug
tool 467
GOBACK statement
RTEREUS run-time option and 70
hardware conditions that cannot be
enabled under 208
options, mapping
mapping STAE to TRAP 93
reusability of an environment when
COBOL is main 69
run-time options specific to
AIXBLD—invoke AM 293, 11
CBLOPTS—specify format of
argument string 18
CBLPSHPOP—control usage of
CICS commands 18

CICS: (continued)
modifier 121, 124
COBOL (continued)

run-time options specific to
(COBOL)
CBLQDA—control QSAM dynamic allocation 59
CHECK—allow for checking errors 22
DEBUG—activate COBOL batch debugging 23
FLOW—control OS/VS COBOL
FLOW output 36
RTEREUS—make first COBOL routine reusable 69
SIMVRD—specify VSAM KSDS 71
UPSII—set UPSII switches 95
severity 0 and 1 conditions,
ERRCOUNT run-time option and 31
space management tuning table, using HEAP run-time option with 38
STOP RUN statement 70
tuning, with HEAP run-time option 38
using VSAM KSDS to simulate variable length relative organization data sets 11
variables, where stored 36, 71, 74
command syntax diagrams xiii
command line parsing of C arguments (ARGPARSE run-time option) 15
specifying whether C redirections are allowed from (REDIR run-time option) 66
specifying whether run-time options can be specified on (EXECOPS run-time option) 32
commands, file parameter 85
compatibility CICS 18
complex numbers complex number math services conjugate of complex
(CEESxCJG) 483
floating complex divide
(CEESxDVDF) 488
floating complex multiply
(CEESxMLT) 498
imaginary part of complex
(CEESxIMG) 493
cond_rep parameter
CEECMBI callable service and 222
CEEGQDT callable service and 321
CEESGL callable service and 454
cond_str parameter 208
cond_token parameter
CEEMGET callable service and 374
CEEMSG callable service and 403
CEENCOD callable service and 408
condition:
active
CEEGPID callable service and 316
dumping information related to 145
condition (continued)
active (continued)
specifying how much information produced for 83
cause code 409
class code 409
enabled 453
getting name of routine that incurred condition (CEEGGRN) 158
examples of 159, 164
getting offset of condition 164
initial allowing the handling of 25
returning initial condition token for (CEEITOK) 357
nested, allowing levels of 24
original 83
promoted 83
providing q_data_token (in CEESGL)
examples of 456, 459
syntax description 453
safe conditions 454
severity CEESGL callable service and 454
ERRCOUNT run-time option and 31
severity levels that cause the debug tool to gain control 84
tolerating a given number of 25
condition handler C signal handlers
CEEMRRCR callable service and 391
CEESGL callable service and 454
HLL semantics
ABPERC run-time option and 9
TRAP run-time option and 93
Language Environment condition
handler 93
PL/1 ON-units
CEESGL callable service and 454
user-written allowing nested conditions in 24
CEESPM callable service and 207
moving the resume cursor from a 391
registering with CEESGL 336
retrieving q_data_token 321
TRAP run-time option and 93
unregistering 347
vector facility and 97
condition handling callable services for
CEE3ABD—terminate enclave with an abend 121, 124, 169
CEE3GRN—get name of routine that incurred condition 158
CEE3GRO—get offset of condition 164
CEESPM—query and modify Language Environment hardware condition enablement 206
CEE3SRP—set resume point 213
CEEDCOD—decompose a condition token 256
condition handling (continued)
callable services for (continued)
CEEESP—retrieve the Language Environment version and platform ID 409
CEEESQDT—retrieve q_data_token 321
CEEERCL—register user condition handler 336
CEEENL—unregister user condition handler 346
CEEITOK—return initial condition token 357
CEEMRCE—move resume cursor explicit 384
CEEMRCR—move resume cursor relative to handle cursor 391
CEENCOD—construct a condition token 407
CEESGL—signal a condition 453
Quick Reference Tables 105, 106
CICS, under 18
depth of conditions allowed 24
dumps, handling of conditions that arise when processing 142
runtime options for
ABPERC 9
DEPTHCONDLMT 24
ERRCOUNT 31
TRAP 93
XUFLOW 100
signaling condition with
CEESGL 453
user-written condition handler allowing nested conditions in
(DEPTHCONDLMT run-time option) 24
moving the resume cursor from a
(CEEMRRCR) 391
registering (CEEERCL) 336
retrieving q_data_token (CEEGQDT) 321
TRAP run-time option and 93
unregistering (CEEENL) 347
vector facility and 97
condition information block
dumping information related to 145
returning initial condition token for
(CEEITOK) 357
examples of 359, 363
Condition Information Block (CIB) 124, 128
condition token altering (CEECMBI) 256, 258
constructing (CEECMBI) 407
examples of 410, 413
decomposing (CEECMBI) 256
dumping information associated with (CEE3DMP) 145
input to CEESGL, using as 454
message corresponding to, getting get, format, and dispatch message
(CEEMSG) 402, 406
get, format, and store message in a buffer (CEEMGET) 374, 380
q_data_token from the ISI, using to retrieve (CEEGQDT) 321
condition token (continued)
examples of 322, 330
returning initial condition token (CEEITOK) 357
examples of 359, 363
SCEESAMP files for 113
country setting
condition_ID portion of condition token 409
conjugate of complex math service
(CEEEXCJG) 483
country setting
COPY statement, in COBOL 113
convert Lilian date to character format
(CEEDATE) 237
equivalent of examples of 239, 243
convert character format to Lilian date
(CEEDAYS) 250
equivalent of examples of 250
convert Lilian date to character format
(CEEDATE) 237
equivalent of examples of 239, 243
control portion of condition token 257,
construction a condition token
control portion of condition token 257, 408
CONTROLLED storage for PL/I 36
convert character format to Lilian date
(CEEDAYS) 250
examples of 250
convert Lilian date to character format
(CEEDATE) 237
equivalent of examples of 239, 243
COPY statement, in COBOL 113
convert cosine math service (CEEEXCOS) 484
cotangent math service (CEEExCTN) 486
COUNTRY run-time option
defaults of country_code parameter 515
relationship to setlocale() 23
country setting
default, querying with CEE3CTY callable service 132
default, setting with CEE3CTY callable service 132
country_code
country codes defaults table 515
in CEE3MCS callable service 181
equivalent of examples of 183, 185
in CEE3MDS callable service 189
equivalent of examples of 191, 192
in CEE3MTS callable service 192
equivalent of examples of 193, 195
in CEEFMDS callable service 284
examples of 285, 287
in CEEFMCT callable service 287
examples of 288, 289
in CEEFMTCM callable service 295
examples of 296, 298
nonexistent country_code, specifying a 22
parameter 515
popping country_code off stack with CEE3CTY 133
pushing prior country_code to top of stack with CEE3CTY 133
querying (CEE3CTY) 132
examples of 135, 137
setting with CEE3CTY callable service 132, 137
with COUNTRY run-time option 22
currency symbol
currency symbol defaults for a given country 515
default, obtaining with CEE3MCS callable service 181
currency symbol (continued)
examples of 183, 185
setting defaults for with CEE3CTY callable service 132
with COUNTRY run-time option 22, 132
currency symbol (continued)
examples of 183, 185
setting defaults for with CEE3CTY callable service 132
with COUNTRY run-time option 22, 132
day and time (continued)
services (continued)
CEESECS—convert timestamp to number of seconds 441
Quick Reference Tables 106
day of week, calculating with
CEE3CTY 272
DB2
POSIX run-time option and 63
DEBUG run-time option
syntax 23
Debug Tool
CEETEST callable service, invoking with 466
giving control to with TEST run-time option 84
invoking with CEE3CTY callable service 466
TEST run-time option and 84
dumping
COBOL batch 23
decimal separator
default, obtaining with CEE3MDS callable service 189
for countries, defaults 515
setting defaults for with CEE3CTY callable service 22, 132
number of seconds for a given country_code 515
DECIMAL-OVERFLOW condition 207, 208
DEPTHCONDLMT run-time option
syntax 25
disability 525
dump
CEE3ABD callable service and abend with clean-up can generate
Language Environment dump 121, 125
abend with clean-up generates only system dump 121, 125
Language Environment dump, requesting
CEE3ABD callable service 145
CEE3MCS callable service 145
examples of 147, 149
syntax description 142, 146
TEST compile-time option and 144
TERMTHDACT run-time option and 79
dynamic storage
allocating additional heap, setting size of 227
initial heap, setting size of 36, 48
callable services for CEE3RPH—set report
heading 203, 206
CEE3RPH—set report
CEE3RPH—create new additional heap 227, 232
CEE3RPH—create new additional heap 227, 232
CEE3ZST—reallocate heap storage 234
dynamic storage (continued)
callable services for (continued)
  CEEEDSHP—discard heap  268, 271
  CEEFRST—free heap storage  298, 301
  CEEGTST—get heap storage  331, 335
Quick Reference Tables  106 initializing (STORAGE run-time option) 75
DYNDFMP run-time option 26
examples (continued)
  CEE3MTS—get default thousands separator  193, 195
  CEE3PRM—query parameter string  196, 198
  CEE3RPH—set report heading  204, 206
  CEE3SPM—query and modify Language Environment hardware condition enablement 209, 210
  CEE3SRC—set enclave return code  213
  CEE3USR—set or query user area fields  215, 218
  CEECM1—store and load message insert data  225, 227
  CEECRHP—create new additional heap  230, 232
  CEECZST—reallocate (change size of) storage  234, 236
  CEEEDATE—convert Lilian date to character format  239, 243
  CEEEDATM—convert seconds to character format  245, 249
  CEEEDCOD—decompose a condition token  259, 261
  CEEEDSHP—discard heap  269, 271
  CEEEDYWK—calculate day of week from Lilian date  273, 277
  CEEFMDA—generate default date format  285, 287
  CEEFMDT—get default date and time format  288, 289
  CEEFMON—format monetary string  293
  CEEFMTM—get default time format  296, 298
  CEEFRST—free heap storage with CEEGTST  300, 301
  CEEFTDS—get default date and time into character string  306
  CEEG1M—get current GMT  310, 313
  CEEG2MT—get offset from GMT  315
  CEEGFID—get default fields 316
  with CEE3GRC and CEE3SRC  150, 157
  CEEHDLR—register user-written condition handler by itself  336
  with CEEHDLR  348, 350
  CEEISEC—convert integers to seconds  354, 356
  CEEITOK—return initial condition token  359, 363
  CEELCNV—query locale numeric conventions  367
  CEELCT—get current local time  372
  CEEMGET—get a message  378, 380
examples (continued)
  CEEMOUT—dispatch a message with CEEMGET  382
  CEEMRC—move resume cursor with CEEHDLR and CEEHDLR  395, 401
  CEEMSG—get, format, and dispatch a message  404, 406
  CEENCOD—construct a condition token  413
  CEENCEN—query century window  415, 417
  CEEQDT—query locale, date, and time conventions  419
  CEEQR2Y—query active locale environment  424
  CEEQSEN—set century window  430, 432
  CESCOL—compare string collation weight  434
  CEESDLG1—calculate logarithm base e  510
  CEESF—convert seconds to integers  438, 441
  CEESFS—convert timestamp to number of seconds  445, 448
  CEESL—set locale operating environment  450
  CEESGL—signal a condition  456, 459
  CEESMOD—perform modular arithmetic  477
  CEESLOG—calculate logarithm base e  477
  CEEST2F—transform string characters into collation weights  461
  CEETEST—invoke debug tool  467, 469
declares in COBOL and PL/I for Language Environment data types  117
math services  476
  exceptions
S/370 interrupt codes  209
EXEC CICS command
  HANDLE ABEND CBLPSHPOP run-time option and 18
  HANDLE AID CBLPSHPOP run-time option and 18
  HANDLE CONDITION CBLPSHPOP run-time option and 18
  POP HANDLE CBLPSHPOP run-time option and 18
  PUSH HANDLE CBLPSHPOP run-time option and 18
EXECOPS run-time option syntax  32
exponent underflow  100, 207, 208
exponential base e math service (CEESXEXP)  491
exponentiation math service (CEESXEXP)  507
external data relationship to ALL31 run-time option 13

F
facility ID
CEENCOD callable service and 408
IGZ severity 0 and 1 conditions and the ERRCOUNT run-time option 31
feedback code
FEEDBACK data type 117
in callable services 117
omitting 117
FILEDEF command for CMS SYSABEND PRINTER 121, 125
SYSUDUMP PRINTER 121, 125
FILEHIST run-time option syntax 33
FILETAG run-time option 34
fixed-overflow condition, in CEE3SPM 207, 208
floating complex divide math service (CEESxVDV) 488
floating complex multiply math service (CEESxMST) 498
FLOW run-time option syntax 36

G
gamma function math service (CEESxGMA) 492
general callable services
CEE3GRC—get enclave return code 149, 157
CEE3PRM—query parameter string 195, 198
CEE3SRC—set enclave return code 212, 213
CEEUSR—set or query user area fields 214, 218
CEERAN0—calculate uniform random numbers 425, 428
CEETEST—invoke debug tool 466, 470
Quick Reference Tables 107
GOBACK statement
RTEREUS run-time option and 70
Greenwich Mean Time (GMT) as seed parameter of CEERAN0 callable service 425
getting offset to local time from CEEGMTO 313
examples of 315
return Lilian date and Lilian seconds (CEEGMT) 309
examples of 310, 313
Gregorian character string returning local time as a (CEELOCT) 370
examples of 372

H
HANDLE ABEND EXEC CICS command
CBLPSHPOP run-time option and 18
handle cursor
moving resume cursor relative to (CEE3MCR) 391
examples of 395
header files
leawi.h (C) callable service declarations and 112
condition token structure and 409
HEAP run-time option
ALL31 run-time option and 12
CEECHRHP and 228
different treatment under CICS 38
STORAGE run-time option and 76
heap storage
allocating
from anywhere heap (ANYHEAP run-time option) 14, 15
from below heap (BELOWHEAP run-time option) 17, 18
from initial heap segment (CEEGTST) 332
of initial heap storage (HEAP run-time option) 36, 48
heap storage, thread-level heap storage (THREADHEAP run-time option) 86
AMODE considerations of 13, 38
callable services for CEE3RPH—set report heading 203
CEECHRHP—create new additional heap 227
CEE3CST—reallocating heap storage 233
CEEDSHP—discard heap storage 268
CEE3FRST—free heap storage 298
CEEGTST—get heap storage 331
Quick Reference Tables 106
clearing after freeing, in STORAGE run-time option 76
discarding an entire heap (CEEDSHP) 268
examples of 269
freeing (CEE3FRST) 298
getting (CEE3GETST) 331
heap element, changing size of (CEE3CST) 233
heap ID
heap ID 0 invalid in CEE3MCR 268
returned by CEECHRHP 228
using to indicate which heap to discard, in CEE3MCR 268
using to receive storage from a given heap, in CEE3GETST 331
heap increment
determining size of, with HEAP run-time option 37
HEAP run-time option and 36, 48
initial heap segment
determining size of (HEAP run-time option) 37
heap storage (continued)
initializing (STORAGE run-time option) 75
managing allocation of (HEAP run-time option) 36, 48
types of variables stored in 36, 48
HEAP64 run-time option 38
HEAPPOLLS64 run-time option 44
Heisei era 522
hyperbolic math services
arctangent (CEESxATH) 480
cosine (CEESxCOS) 485
sine (CEESxSIN) 503
tangent (CEESxTAN) 506
hyperbolic arctangent math service (CEESxATH) 480
hyperbolic cosine math service (CEESxCOS) 485
hyperbolic sine math service (CEESxSIN) 503
hyperbolic tangent math service (CEESxTAN) 507

I
I/O
BELOWHEAP run-time option and 17
IGZ Facility ID
ERRCOUNT run-time option and 31
imaginary part of complex math service (CEESxIMG) 493
IMS (Information Management System)
PLIST run-time option and 61
POSIX run-time option and 62, 91
specifying a C application is running under 28, 29
include statement, in C and PL/I 113
INFOMSGFILTER run-time option 46
INFOMSGFILTER—eliminates unwanted informational messages 46
initial heap
allocating storage from (HEAP run-time option, CEE3GETST) 36, 48, 332
restrictions against discarding 268
initial heap segment
determining size of (HEAP run-time option) 36, 48
reallocating (changing size of) 232
initializing storage
using options of CEECHRHP callable service 228
using STORAGE run-time option 75, 78
input/output
See I/O
INQPCOPN run-time option syntax 46
insert data
Language Environment-generated 408
user-created
cannot use CEE3MOUT callable service to create 381
insert data (continued)
storing and loading 222
INSPPREF preference file 86
instance specific information (ISI) CEECDXOD callable service and 257
creating, when building a condition token 408
insert data is part of 408
maintaining a number of 55
overwriting 223
retrieving q_data_token from (CEEGQDT) 321
examples of 322, 330
storing address of message insert data in 222
using CEESGL callable service to create 453, 454
examples of 456, 459
integers
converting Lilian seconds to (CEESECI) 436
examples of 441
converting to Lilian seconds (CEEISEC) 351, 354, 356
INTERUPT run-time option debug tool and 47
syntax 47
TEST run-time option and 48
intrinsic functions, compatibility with CEELOCT callable service 370
invoking Debug Tool
with CEETEST callable service 466, 470
with TEST run-time option 84
IOHEAP64 run-time option 48

J
Japanese eras 522

K
keyboard 525

L
language-specific condition handlers, use with XUFLQ run-time option 98
leawi file, description of 113
library
stack storage
specifying size of (LIBSTACK run-time option) 51
LIBSTACK run-time option syntax 51
Lilian date
calculate day of week from (CEEDYWK) 272
convert date to (CEEDAYS) 218, 250
convert output_seconds to (CEEIFSEC) 352
convert to character format (CEEDATE) 237
Lilian date (continued)
get current local date or time as a (CEELOCT) 370
get GMT as a (CEEGMT) 309
using as input to CEESCEI callable service 436
local time
getting (CEELOCT) 370
locale services
CEEFMON—format monetary string 290
CEEFTDS—format date and time into character string 303
CEEQLCNV—query locale numeric conventions 363
CEEQDTC—return locale date and time 417
CEEQRYL—query active locale environment 423
CEESCOL—compare string collation weight 432
CEESETL—set locale operating environment 448
CEESTXF—transform string character into collation weight 459
log gamma math service (CEESxLGM) 494
logarithm base 10 math service (CEESxLG1) 495
equations using 510
logarithm base 2 math service (CEESxLG2) 496
logarithm base e math service (CEESxLOG) 497
equations using 510
logarithm routines
base 10 (CEESxLG1) 495
base 2 (CEESxLG2) 496
base e (CEESxLOG) 497
log gamma (CEESxLGM) 494
math services (continued)
hyperbolic cosine (CEESxCOSH) 485
hyperbolic sine (CEESxSINH) 503
hyperbolic tangent (CEESxTANH) 506
imaginary part of complex (CEESxIMG) 493
log gamma (CEESxLGM) 494
logarithm base 10 (CEESxLG1) 495
logarithm base 2 (CEESxLG2) 496
logarithm base e (CEESxLOG) 497
modular arithmetic (CEESxMOD) 498
nearest integer (CEESxNIN) 499
nearest whole number (CEESxNWN) 500
positive difference (CEESxDIM) 487
sine (CEESxSIN) 502
square root (CEESxSQRT) 504
tangent (CEESxTAN) 505
transfer of sign (CEESxSGN) 501
truncation (CEESxINT) 493
Meiji era 522
message
get, format, and dispatch a message (CEEMSG) 402, 404, 406
get, format, and store message in a buffer (CEEMGET) 374, 380
obtaining 374, 402
truncated 375
message file
pre-Language Environment options to Language Environment options 101
relationship to CBLQDA run-time option 19
message handling
nested conditions and 55
quick reference of callable services for 109
MinKow era 523
modular arithmetic math service (CEESxMOD) 499
examples using 510
MSGFILE run-time option
different treatment under CICS 54
RPTOPTS options report and 52
RPTSTG storage report and 52
MSQ run-time option
relationship to Instance Specific Information (ISIs) 55
syntax 55
MTF (Multitasking Facility)
allocating heap storage under (HEAP run-time option) 38
multithreading
RPTSTG run-time option and 68

N
national language
querying (CEESLNQ) 174
setting (NATLANG run-time option, CEE3LNG) 56, 174
national language support (NLS)
default values for a specified country 515
quick reference of callable services for 107, 109
Index 547
national language support (NLS)
(continued)
specifying national language
(NATLANG run-time option) 56
natural log math service
(CEESxLOG) 497
nearest integer math service
(CEESxNIN) 500
nearest whole number math service
(CEESxNWN) 501
nested condition
getting name of routine that incurred
a condition (CEE3GRN) 158
limiting (DEPTHCONDLMT run-time
option) 24
MSGQ run-time options and 55
Notices 529
OCSTATUS run-time option
syntax 57
offset of condition, getting
(CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
offset of condition, getting
CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
offset of condition, getting
CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
offset of condition, getting
CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
offset of condition, getting
CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
offset of condition, getting
CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
offset of condition, getting
CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
offset of condition, getting
CEE3GRO) 164, 165, 214
omitted parameter
how C indicates omission 114
ONCHAR built-in function, in
CEE3DMP 145
ONCOUNT built-in function, in
CEE3DMP 145
ONFILE built-in function, in
CEE3DMP 145
ONKEY built-in function, in
CEE3DMP 145
ONSOURCE built-in function, in
CEE3DMP 145
options report, generating (RPTOPTS
run-time option) 67
q_data_token, in CEESGL (continued)
retrieving from the ISI
(CEEQQT) 321
QUERY function
using to check the current country
setting, in CEE3CTY 133
using to check the current national
language, in CEE3LNG 175
using to check the enablement of
hardware conditions, in
CEE3SPM 207
Quick Reference Tables 3

R
random numbers
generation of (CEERAN0) 425
RDRUNIT run-time option
syntax 65
reason code
CEE3MP callable service and 145
dumps and 145
RECPAD run-time option
syntax 65
REDIR run-time option
syntax description 66
redirections
of stderr, stdout and stdin output 54,
66
REDIR run-time option and 66
register
save area, as component of
dsa_alloc_value 76
report
generating options report (RPTOPTS
run-time option) 67
resume
cursor
moving (CEEMRCR) 391
setting resume point (CEE3SRP) 213
reusability of an environment 69
routine that incurred condition, getting
(CEE3GRN) 158, 159, 164
RPTOPTS run-time option
relationship to MSGFILE run-time
option 67
sample options report generated by
67
syntax 67
RPTSTG run-time option
CEE3GTST and 332
storage report generated by
setting heading for 203
RTERUS run-time option
CICS ignores this option 70
syntax 69
run-time options
ABPERC—percolate an abend 9
AIXBLD—invoke AMS for
COBOL 11
ALL31—indicate whether application
runs in AMODE(31) 12
ANYHEAP—control unrestricted
library heap storage 14
ARGPARSE—specify whether
arguments are parsed 15
run-time options (continued)
AUTOTASK—specify whether Fortran
MTF is to be used 16
BELOWHEAP—control library heap
storage below 16M 17
CBOPTS—specify format of COBOL
argument 18
CBLPSHPPOP—control CICS
commads 18
CBLQDA—control COBOL
QSAM 19
CHECK—detect checking errors 22
COUNTRY—specify default date/time
formats 22
DEBUG—activate COBOL batch
debugging 23
DEPTHCOND—limit extent of
nested conditions 24
ENV—specify operating environment
for C application 29
ERRCOUNT—specify number of
errors allowed 31
ERRUNIT—specify unit number to
which error information is
directed 32
EXECOPS—let run-time options be
specified on command line 32
FILEHIST—specify whether to allow a
file definition to be changed at run
time 33
FILETAG—specify whether to allow
AUTOTAG / AUTOCVT 34
FLOW—control FLOW output for
OS/VS COBOL 36
HEAP—control allocation of
heaps 36, 48
HEAP64—controls allocation of user
heap storage 38
HEAPPOLL64 — controls optional
user heap storage management
algorithm 44
INQPCOPN—control value in
OPENED specifier of INQUIRE by
unit statement 46
INTERRUPT—cause attentions to be
recognized by Language
Environment 47
IOHEAP64 — controls allocation of
I/O heap storage 48
LIBSTACK—control library stack
storage 51
MSGQ—specify number of ISI blocks
allocated 55
NATLANG—specify national
language 56
OCSTATUS—control checking of file
existence and whether file deletion
occurs 57
PC—control whether Fortran status
common blocks are shared among
load modules 60
PLIST—specify format of C
arguments 61
PLITASKCOUNT—control the
maximum number of active
tasks 62

Q
q_data_token, in CEESGL (continued)
creating 454
run-time options (continued)
- PRTUNIT—specifies unit number used for PRINT and WRITE statements 64
- PUNUNIT—specifies unit number used for PUNCH statements 64
- Quick Reference Tables 3, 5
- RDRUNIT—specifies unit number used for READ statements 65
- RECPAD—specifies whether a formatted input record is padded with blanks 65
- REDIR—specifies redirections for C output 66
- report of options specified generating heading for 203 language report is written in 56, 67, 68
- not generated if application abends 67
- ex-sample of 67
- RPTOPTS—generate a report of run-time options used 67
- RTEFEUS—initialize a reusable COBOL environment 69
- SIMVRD—specify VSAM KSDS for COBOL 71
- STACK—allocate stack storage 71, 74
- STORAGE—control storage 75
- TRAP—control storage 75
- TRAP—control thread storage 75
- TRAP—indicate enabled hardware conditions with 207
- changing the COUNTRY setting with 133
- changing the enablement of hardware conditions with 207
- changing the national language with 174, 175
- severity of a condition
  - CEESGL and 454
  - severity (continued)
    - of a condition (continued)
      - ERRCOUNT run-time option and 31
      - severity levels that cause the debug tool to gain control 84
      - shortcut keys 525
      - Showa era 522
      - SIGNIFICANCE condition 207, 208
      - SIMVRD run-time option syntax 71
      - sine math service (CEESxSIN) 502
      - slash (/) NOEXECOPS alters behavior of 32
      - specifying in parameters of TEST run-time option 85
      - SPIE run-time option 93
      - square root math service (CEESxSQRT) 505
      - SRANGE option for VS COBOL II 22
      - stack frame
        - CEE3DMP callable service and 145
        - library, allocating (LIBSTACK run-time option) 51
      - storage
        - ALL31 run-time option and 12
        - allocating (STACK run-time option) 71, 74
        - initializing (STACK run-time option) 75
        - RPTSTG run-time option and 69
        - setting initial stack segment (STACK run-time option) 71, 74
        - threads and 69
        - user, allocating (STACK run-time option) 71, 74
      - STACK run-time option
        - syntax 71, 74
        - using with RPTSTG to tune the stack 69
      - STAE
        - C options 93
        - VS COBOL II options 93
      - stderr
        - MSGFILE run-time option and 54
        - REDIR run-time option and 66
        - redirecting output from 66
        - stdout 66
        - MSGFILE run-time option and 66
        - REDIR run-time option and 66
        - redirecting output from 66
        - stderr 66
        - MSGFILE run-time option and 66
        - REDIR run-time option and 66
        - redirecting output from 66
        - storage additional heaps, creating new (CEECRHP) 227
        - discarding an entire heap (CEEDSHP) 268
        - freeing
          - discarding an entire heap (CEEDSHP) 268, 271
          - freeing additional heap (CEEFIRST) 298, 301
  - severity (continued)
    - of a condition (continued)
      - ERRCOUNT run-time option and 31
      - severity levels that cause the debug tool to gain control 84
      - shortcut keys 525
      - Showa era 522
      - SIGNIFICANCE condition 207, 208
      - SIMVRD run-time option syntax 71
      - sine math service (CEESxSIN) 502
      - slash (/) NOEXECOPS alters behavior of 32
      - specifying in parameters of TEST run-time option 85
      - SPIE run-time option 93
      - square root math service (CEESxSQRT) 505
      - SRANGE option for VS COBOL II 22
      - stack frame
        - CEE3DMP callable service and 145
        - library, allocating (LIBSTACK run-time option) 51
      - storage
        - ALL31 run-time option and 12
        - allocating (STACK run-time option) 71, 74
        - initializing (STACK run-time option) 75
        - RPTSTG run-time option and 69
        - setting initial stack segment (STACK run-time option) 71, 74
        - threads and 69
        - user, allocating (STACK run-time option) 71, 74
      - STACK run-time option
        - syntax 71, 74
        - using with RPTSTG to tune the stack 69
      - STAE
        - C options 93
        - VS COBOL II options 93
      - stderr
        - MSGFILE run-time option and 54
        - REDIR run-time option and 66
        - redirecting output from 66
        - stdout 66
        - MSGFILE run-time option and 66
        - REDIR run-time option and 66
        - redirecting output from 66
        - stderr 66
        - MSGFILE run-time option and 66
        - REDIR run-time option and 66
        - redirecting output from 66
        - storage additional heaps, creating new (CEECRHP) 227
        - discarding an entire heap (CEEDSHP) 268
        - freeing
          - discarding an entire heap (CEEDSHP) 268, 271
          - freeing additional heap (CEEFIRST) 298, 301
  - storage (continued)
    - getting
      - heap storage (CEEFIRST) 331
      - initializing (storage run-time option) 75
      - options for
        - ANYHEAP—control library heap 14
        - BELOWHEAP—control library heap below 16M 17
        - HEAP—control initial heap 36, 48
        - LIBSTACK—control library stack storage 51
        - STACK—control thread stack storage 71, 74
        - STORAGE—control initial heap or stack 75
      - report
        - language written in 56, 67, 68
        - not generated if application abends 68
        - setting heading for 203
        - STACK run-time option and 69
        - services
          - CEE3RPH—set report heading 203
          - CEECRHP—create new additional heap 227
          - CEECZST—reallocate (change size of) storage 233
          - CEEDSHP—discard heap 268
          - CEEFRST—free heap storage 299
          - CEEOGMT—get heap storage 331
          - thread-level heap, controlling (THREADHEAP run-time option) 86
        - tuning
          - additional heap, setting size of (CEECRHP) 227
          - anywhere heap, setting size of (ANYHEAP) 14
          - below heap, setting size of (BELOWHEAP) 17
          - initial heap, setting size of (HEAP) 36, 48
          - initial stack segment, setting size of (STACK) 71, 74
          - library stack storage, setting size of (LIBSTACK) 51
        - with CEEGMST 331
        - with STORAGE run-time option 75
      - STORAGE run-time option
        - CEECZST callable service and 233
        - CEEFRST callable service and 299
        - syntax 75
      - straight double quote (") initializing storage with 76, 78
        - specifying in parameters of TEST run-time option 85, 86
      - straight single quote (') initializing storage with 76, 78
        - specifying in parameters of TEST run-time option 85, 86
      - symbol
        - table, generated by CEE3DMP 144

---

S

safe condition 454
SCESAMP sample library declaration files in 113
sending product messages to a file (CEEMSG) 402
examples of 404, 406
sending user-defined message string to file (CEEMOUT) 380
examples of 382, 384
SET function
changing the COUNTRY setting with 133
changing the enablement of hardware conditions with 207
changing the national language with 174, 175
severity of a condition
CEESGL and 454
syntax diagrams
how to read xiii
SYSABEND PRINTER 121, 125
SYSOUT
  default destinations of MSGFILE
    run-time option 53
SYSUDUMP PRINTER 121, 125

T
  Taisho era 522
Taiwan era 523
tangent math service (CEESxTAN) 505
task
  controlling number of tasks under
    MTF (PLITASKCOUNT option) 83
termination
  enclave
    CEE3ABD and 121, 124, 169
termination imminent step
  TERMTHDACT run-time option
    and 83
TERMTHDACT run-time option
  CEE3DMP and 83
  CESE transient data queue and 83
different treatment under CICS
  83
TEST compile-time option 144
TEST run-time option
  syntax 84
thousands separator
  obtaining default of (CEE3CTY) 132
  setting defaults for (COUNTRY) 22
thread
  multiple
    storage report for (RPTSTG
      run-time option) 69
THREADHEAP—control the allocation of
  thread-level heap storage
syntax 86
time, getting local (CEELOCT) 370
timestamp 243, 441
TRACE run-time option
  syntax 91
trace, generating a 79
  transfer of sign math service
    (CEESxSGN) 501
TRAP run-time option
  ABPERC run-time option and 9
  syntax 93
truncation math service (CEESxINT) 494

U
  UNDERFLOW condition 100, 207, 208
UPSII run-time option
  syntax 95
USE FOR DEBUGGING declarative 23
user
  area fields 212, 214, 215
heap (initial heap)
  allocating storage from 36, 48, 332
  restrictions against discarding 268
stack, controlling (STACK run-time option) 69
user-written condition handler
  CEE3SPM callable service and 207
  CEEMRKC callable service and 391
  registering with CEEHDLR callable
    service 336
  unregistering 347
  VCTRSAVE run-time option and 97
  USRHDLR run-time option
    syntax 96

V
  valid condition 453
VCTRSAVE run-time option
  syntax 97
vector facility 97
VSAM
  KSDS 11, 71
  RRDS 11

W
  WITH DEBUGGING MODE clause for
    COBOL 24

X
  XPLINK run-time option 98
  XPLink 98
  XUFLOW run-time option
    syntax 100

Z
  z/OS Basic Skills information center xv