Seventh Edition, December 2004

This is a minor revision of SA22-7562-05.

This edition applies to Language Environment® in Version 1 Release 6 of z/OS (5694-A01), Version 1 Release 6 of z/OS.e™ (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. A form for readers’ comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcls@us.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
• Title and order number of this document
• Page number or topic related to your comment

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

© Copyright International Business Machines Corporation 1991, 2004. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Figures. ................................................................. xi

Tables ................................................................. xv

About this document .............................................. xvii
Using your documentation ....................................... xviii
Where to find more information ................................. xix
   Accessing z/OS licensed documents on the Internet .... xix
   Using LookAt to look up message explanations ........ xx
   Information updates on the web .............................. xx

Summary of Changes .............................................. xxi

Part 1. Language Environment Run-Time Options. ........ 1

Chapter 1. Language Environment run-time option reference tables and general information ................................ 3
   Quick reference tables for AMODE 31 run-time options .... 3
   Quick reference tables for AMODE 64 run-time options .... 5
   How to read syntax diagrams .................................. 6
      Symbols .......................................................... 7
      Syntax items ................................................... 7
      Syntax examples ............................................... 7
   How to specify run-time options ............................. 9
   Propagating run-time options with spawn and exec .... 9

Chapter 2. Using the Language Environment run-time options ........ 11
   ABPERC .............................................................. 11
      z/OS UNIX consideration ................................... 12
      Usage notes .................................................... 12
   ABTERMENC .......................................................... 12
      z/OS UNIX consideration ................................... 13
      Usage notes .................................................... 13
      For more information ....................................... 13
   AIXBLD (COBOL Only) ............................................ 13
      z/OS UNIX consideration ................................... 14
      Performance consideration ................................ 14
      For more information ....................................... 14
   ALL31 ............................................................... 14
      z/OS UNIX considerations ................................ 15
      Usage notes .................................................... 15
      Performance Consideration ................................ 16
      For More Information ....................................... 16
   ANYHEAP ............................................................ 16
      CICS considerations .......................................... 17
      z/OS UNIX considerations ................................ 17
      Performance consideration ................................ 17
      For more information ....................................... 17
   ARGPARSE | NOARGPARSE (C only) ....................... 17
      Usage notes .................................................... 18
      For more information ....................................... 18
   AUTOTASK | NOAUTOTASK (Fortran only) ............... 18
   BELOWHEAP ......................................................... 19

© Copyright IBM Corp. 1991, 2004
<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance consideration</td>
<td>61</td>
</tr>
<tr>
<td>For more information</td>
<td>61</td>
</tr>
<tr>
<td>RPTSTG</td>
<td>61</td>
</tr>
<tr>
<td>CICS consideration</td>
<td>62</td>
</tr>
<tr>
<td>z/OS UNIX consideration</td>
<td>62</td>
</tr>
<tr>
<td>Usage notes</td>
<td>62</td>
</tr>
<tr>
<td>Performance consideration</td>
<td>62</td>
</tr>
<tr>
<td>For more information</td>
<td>62</td>
</tr>
<tr>
<td>RTEREUS (COBOL only)</td>
<td>63</td>
</tr>
<tr>
<td>Usage notes</td>
<td>63</td>
</tr>
<tr>
<td>Performance consideration</td>
<td>64</td>
</tr>
<tr>
<td>For more information</td>
<td>64</td>
</tr>
<tr>
<td>SIMVRD (COBOL only)</td>
<td>64</td>
</tr>
<tr>
<td>For more information</td>
<td>65</td>
</tr>
<tr>
<td>STACK</td>
<td>65</td>
</tr>
<tr>
<td>CICS consideration</td>
<td>66</td>
</tr>
<tr>
<td>z/OS UNIX consideration</td>
<td>66</td>
</tr>
<tr>
<td>Usage notes</td>
<td>66</td>
</tr>
<tr>
<td>Performance consideration</td>
<td>67</td>
</tr>
<tr>
<td>For more information</td>
<td>67</td>
</tr>
<tr>
<td>STACK64 (AMODE 64 only)</td>
<td>68</td>
</tr>
<tr>
<td>Usage notes</td>
<td>68</td>
</tr>
<tr>
<td>Performance consideration</td>
<td>68</td>
</tr>
<tr>
<td>For more information</td>
<td>68</td>
</tr>
<tr>
<td>STORAGE</td>
<td>68</td>
</tr>
<tr>
<td>CICS consideration</td>
<td>71</td>
</tr>
<tr>
<td>z/OS UNIX consideration</td>
<td>71</td>
</tr>
<tr>
<td>Usage notes</td>
<td>71</td>
</tr>
<tr>
<td>Performance consideration</td>
<td>71</td>
</tr>
<tr>
<td>TERMTHDACT</td>
<td>71</td>
</tr>
<tr>
<td>CICS consideration</td>
<td>73</td>
</tr>
<tr>
<td>Usage notes</td>
<td>76</td>
</tr>
<tr>
<td>For more information</td>
<td>77</td>
</tr>
<tr>
<td>TEST</td>
<td>NOTEST</td>
</tr>
<tr>
<td>Usage notes</td>
<td>79</td>
</tr>
<tr>
<td>Performance consideration</td>
<td>79</td>
</tr>
<tr>
<td>For more information</td>
<td>79</td>
</tr>
<tr>
<td>THREADHEAP</td>
<td>79</td>
</tr>
<tr>
<td>CICS consideration</td>
<td>80</td>
</tr>
<tr>
<td>Usage notes</td>
<td>80</td>
</tr>
<tr>
<td>THREADSTACK</td>
<td>80</td>
</tr>
<tr>
<td>Usage notes</td>
<td>83</td>
</tr>
<tr>
<td>THREADSTACK64 (AMODE 64 only)</td>
<td>83</td>
</tr>
<tr>
<td>Usage notes</td>
<td>84</td>
</tr>
<tr>
<td>For more information</td>
<td>84</td>
</tr>
<tr>
<td>TRACE</td>
<td>84</td>
</tr>
<tr>
<td>Usage notes</td>
<td>85</td>
</tr>
<tr>
<td>For more information</td>
<td>85</td>
</tr>
<tr>
<td>TRAP</td>
<td>86</td>
</tr>
<tr>
<td>CICS consideration</td>
<td>87</td>
</tr>
<tr>
<td>Usage notes</td>
<td>87</td>
</tr>
<tr>
<td>For more information</td>
<td>88</td>
</tr>
<tr>
<td>UPSI (COBOL only)</td>
<td>88</td>
</tr>
<tr>
<td>CICS consideration</td>
<td>89</td>
</tr>
<tr>
<td>Usage notes</td>
<td>89</td>
</tr>
<tr>
<td>For more information</td>
<td>89</td>
</tr>
</tbody>
</table>
### Part 2. Language Environment callable services

#### Chapter 3. Quick reference tables for Language Environment services

Bit manipulation routines ............................................. 97  
Condition handling callable services ............................... 97  
Date and time callable services .................................... 98  
Dynamic storage callable services .................................. 99  
General callable services ............................................. 99  
Initialization and termination services ............................ 99  
Locale callable services ............................................. 99  
Math services .......................................................... 100  
Message handling callable services ................................ 101  
National Language Support callable services ...................... 101  

#### Chapter 4. Language Environment callable services

Locating callable service information ................................ 103  
General usage notes for callable services .......................... 104  
Invoking callable services ............................................ 105  
Header, copy, or include files ........................................ 105  
Sample programs ....................................................... 106  
C/C++ syntax ........................................................... 107  
COBOL syntax .......................................................... 107  
PL/I syntax .............................................................. 108  
Parameter list for invoking callable services ....................... 109  
Data type definitions .................................................. 110  
C/C++ data type definitions ......................................... 110  
COBOL data type definitions ....................................... 111  
PL/I data type definitions .......................................... 112  
Callable services ..................................................... 113  
CEE3ABD—Terminate enclave with an abend ...................... 113  
CEE3CIB—Return pointer to condition information block .......... 116  
CEE3CTY—Set default country ...................................... 120  
CEE3DMP—Generate dump .............................................. 127  
CEE3GRC—Get the enclave return code ............................ 135  
CEE3GRN—Get name of routine that incurred condition ........... 148  
CEE3GRO—Get offset of condition .................................. 156  
CEE3LANG—Set national language .................................... 163  
CEE3MCS—Get default currency symbol ............................ 172  
CEE3MDS—Get default decimal separator ........................... 175  
CEE3MTS—Get Default thousands separator ......................... 179  
CEE3PRM—Query parameter string ................................... 183  
CEE3RPH—Set report heading ........................................ 186
Chapter 6. Language Environment math services

Call interface to math services

- Parameter types: parm1 Type and parm2 type
- Feedback code parameter (fc)
- Language-specific built-in math services

Sample calls to math services

- C/C++ call to CEESLOG—logarithm base e
- COBOL call to CEESSLOG—logarithm base e
- PL/I call to CEESSLOG—logarithm base e

Math services

- CEESxABS—Absolute value
- CEESxACS—Arccosine
- CEESxASN—Arcsine
- CEESxATH—Hyperbolic arctangent
- CEESxATN—Arctangent
- CEESxAT2—Arctangent2
- CEESxCJG—Conjugate of complex
- CEESxCOS—Cosine
- CEESxCSH—Hyperbolic cosine
- CEESxCSE—Cotangent
- CEESxDIM—Positive difference
- CEESxDVD—Floating-point complex divide
- CEESxERC—Error function complement
- CEESxERF—Error function
- CEESxEXP—Exponential base e
- CEESxGMA—Gamma function
- CEESxIMG—Imaginary part of complex
- CEESxINT—Truncation
- CEESxLG1—Logarithm base 10
- CEESxLG2—Logarithm base 2
- CEESxLOG—Logarithm base e
- CEESxMLT—Floating-point complex multiply
- CEESxMOD—Modular arithmetic
- CEESxNIN—Nearest integer
- CEESxNWN—Nearest whole number
- CEESxSGN—Transfer of sign
- CEESxSIN—Sine
- CEESxSNH—Hyperbolic sine
- CEESxSQT—Square root
- CEESxTAN—Tangent
- CEESxTNH—Hyperbolic tangent
- CEESxXPx—Exponentiation

Examples of math services

- C/C++ math service example
- COBOL math service examples
- PL/I math service examples
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Effect of DEPTHCONDLMT(3) on Condition Handling</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>An Invalid COBOL CALL that Omits the fc Parameter</td>
<td>104</td>
</tr>
<tr>
<td>3</td>
<td>An Invalid PL/I CALL that Omits the fc Parameter</td>
<td>104</td>
</tr>
<tr>
<td>4</td>
<td>Valid COBOL CALLs that Use the Optional fc Parameter.</td>
<td>104</td>
</tr>
<tr>
<td>5</td>
<td>Valid PL/I CALLs that Use the Optional fc Parameter.</td>
<td>105</td>
</tr>
<tr>
<td>6</td>
<td>Sample Callable Services Invocation Syntax for C/C++</td>
<td>107</td>
</tr>
<tr>
<td>7</td>
<td>Sample Callable Services Invocation Syntax for COBOL</td>
<td>108</td>
</tr>
<tr>
<td>8</td>
<td>Sample Callable Services Invocation Syntax for PL/I</td>
<td>109</td>
</tr>
<tr>
<td>9</td>
<td>C/C++ Example of CEE3ABD</td>
<td>114</td>
</tr>
<tr>
<td>10</td>
<td>COBOL Example of CEE3ABD</td>
<td>115</td>
</tr>
<tr>
<td>11</td>
<td>PL/I Example of CEE3ABD</td>
<td>115</td>
</tr>
<tr>
<td>12</td>
<td>C/C++ Example of EDC3CIB</td>
<td>117</td>
</tr>
<tr>
<td>13</td>
<td>COBOL Example of CEE3CIB</td>
<td>119</td>
</tr>
<tr>
<td>14</td>
<td>PL/I Example of CEE3CIB</td>
<td>120</td>
</tr>
<tr>
<td>15</td>
<td>C/C++ Example of CEE3CTY</td>
<td>123</td>
</tr>
<tr>
<td>16</td>
<td>COBOL Example of CEE3CTY</td>
<td>124</td>
</tr>
<tr>
<td>17</td>
<td>PL/I Example of CEE3CTY</td>
<td>126</td>
</tr>
<tr>
<td>18</td>
<td>C/C++ Example of CEE3DMP</td>
<td>133</td>
</tr>
<tr>
<td>19</td>
<td>COBOL Example of CEE3DMP</td>
<td>134</td>
</tr>
<tr>
<td>20</td>
<td>PL/I Example of CEE3DMP</td>
<td>135</td>
</tr>
<tr>
<td>21</td>
<td>C/C++ main() Routine that Calls CEEHDLR and CEE3GRC</td>
<td>137</td>
</tr>
<tr>
<td>22</td>
<td>C/C++ User-Written Condition Handler that Sets a User Enclave Return Code.</td>
<td>139</td>
</tr>
<tr>
<td>23</td>
<td>C/C++ Subroutine that Generates the Divide-by-Zero Condition</td>
<td>140</td>
</tr>
<tr>
<td>24</td>
<td>COBOL Main Routine that Calls CEEHDLR and CEE3GRC</td>
<td>141</td>
</tr>
<tr>
<td>25</td>
<td>COBOL Condition Handler that Sets a User Enclave Return Code and Resumes when a Divide-by-Zero Condition Occurs</td>
<td>143</td>
</tr>
<tr>
<td>26</td>
<td>COBOL Subroutine that Generates a Divide-by-Zero</td>
<td>146</td>
</tr>
<tr>
<td>27</td>
<td>PL/I Example that Sets and Retrieves the User Enclave Return Code when a Divide-by-Zero is Generated</td>
<td>147</td>
</tr>
<tr>
<td>28</td>
<td>C/C++ Example of CEE3GRN</td>
<td>150</td>
</tr>
<tr>
<td>29</td>
<td>COBOL Example of CEE3GRN</td>
<td>152</td>
</tr>
<tr>
<td>30</td>
<td>PL/I Example of CEE3GRN</td>
<td>156</td>
</tr>
<tr>
<td>31</td>
<td>COBOL Example of CEE3GRO</td>
<td>158</td>
</tr>
<tr>
<td>32</td>
<td>PL/I Example of CEE3GRO</td>
<td>161</td>
</tr>
<tr>
<td>33</td>
<td>C/C++ Example of CEE3LNG</td>
<td>167</td>
</tr>
<tr>
<td>34</td>
<td>COBOL Example of CEE3LNG</td>
<td>168</td>
</tr>
<tr>
<td>35</td>
<td>PL/I Example of CEE3LNG</td>
<td>170</td>
</tr>
<tr>
<td>36</td>
<td>C/C++ Example of CEE3MCS</td>
<td>173</td>
</tr>
<tr>
<td>37</td>
<td>COBOL Example of CEE3MCS</td>
<td>174</td>
</tr>
<tr>
<td>38</td>
<td>PL/I Example of CEE3MCS</td>
<td>175</td>
</tr>
<tr>
<td>39</td>
<td>C/C++ Example of CEE3MDS</td>
<td>177</td>
</tr>
<tr>
<td>40</td>
<td>COBOL Example of CEE3MDS</td>
<td>178</td>
</tr>
<tr>
<td>41</td>
<td>PL/I Example of CEE3MDS</td>
<td>179</td>
</tr>
<tr>
<td>42</td>
<td>C/C++ Example of CEE3MTS</td>
<td>181</td>
</tr>
<tr>
<td>43</td>
<td>COBOL Example of CEE3MTS</td>
<td>182</td>
</tr>
<tr>
<td>44</td>
<td>PL/I Example of CEE3MTS</td>
<td>183</td>
</tr>
<tr>
<td>45</td>
<td>C/C++ Example of CEE3PRM</td>
<td>184</td>
</tr>
<tr>
<td>46</td>
<td>COBOL Example of CEE3PRM</td>
<td>185</td>
</tr>
<tr>
<td>47</td>
<td>PL/I Example of CEE3PRM</td>
<td>186</td>
</tr>
<tr>
<td>48</td>
<td>C/C++ Example of CEE3RPH</td>
<td>187</td>
</tr>
<tr>
<td>49</td>
<td>COBOL Example of CEE3RPH</td>
<td>188</td>
</tr>
<tr>
<td>50</td>
<td>PL/I Example of CEE3RPH</td>
<td>189</td>
</tr>
<tr>
<td>51</td>
<td>C/C++ Example of CEE3SPM</td>
<td>193</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1991, 2004
<table>
<thead>
<tr>
<th>Page</th>
<th>Example Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>COBOL Example of CEE3SPM</td>
<td>194</td>
</tr>
<tr>
<td>53</td>
<td>PL/I Example of CEE3SPM</td>
<td>195</td>
</tr>
<tr>
<td>54</td>
<td>C/C++ Example of CEE3USR</td>
<td>200</td>
</tr>
<tr>
<td>55</td>
<td>COBOL Example of CEE3USR</td>
<td>201</td>
</tr>
<tr>
<td>56</td>
<td>PL/I Example of CEE3USR</td>
<td>202</td>
</tr>
<tr>
<td>57</td>
<td>COBOL Example of CEECBLDY</td>
<td>207</td>
</tr>
<tr>
<td>58</td>
<td>C/C++ Example of CEECMI</td>
<td>210</td>
</tr>
<tr>
<td>59</td>
<td>COBOL Example of CEECMI</td>
<td>212</td>
</tr>
<tr>
<td>60</td>
<td>PL/I Example</td>
<td>214</td>
</tr>
<tr>
<td>61</td>
<td>C/C++ Example of CECCRHNP</td>
<td>218</td>
</tr>
<tr>
<td>62</td>
<td>COBOL Example of CECCRHNP</td>
<td>219</td>
</tr>
<tr>
<td>63</td>
<td>PL/I Example of CECCRHNP</td>
<td>220</td>
</tr>
<tr>
<td>64</td>
<td>C/C++ Example of CEECZST</td>
<td>223</td>
</tr>
<tr>
<td>65</td>
<td>COBOL Example of CEECZST</td>
<td>224</td>
</tr>
<tr>
<td>66</td>
<td>PL/I Example of CEECZST</td>
<td>226</td>
</tr>
<tr>
<td>67</td>
<td>C/C++ Example of CEEDATE</td>
<td>230</td>
</tr>
<tr>
<td>68</td>
<td>COBOL Example of CEEDATE</td>
<td>230</td>
</tr>
<tr>
<td>69</td>
<td>PL/I Example of CEEDATE</td>
<td>233</td>
</tr>
<tr>
<td>70</td>
<td>C/C++ Example of CEEDATM</td>
<td>237</td>
</tr>
<tr>
<td>71</td>
<td>COBOL Example of CEEDATM</td>
<td>238</td>
</tr>
<tr>
<td>72</td>
<td>PL/I Example of CEEDATM</td>
<td>241</td>
</tr>
<tr>
<td>73</td>
<td>C/C++ Example of CEEDAYS</td>
<td>245</td>
</tr>
<tr>
<td>74</td>
<td>COBOL Example of CEEDAYS</td>
<td>246</td>
</tr>
<tr>
<td>75</td>
<td>PL/I Example of CEEDAYS</td>
<td>248</td>
</tr>
<tr>
<td>76</td>
<td>C/C++ Example of CEEDCOD</td>
<td>252</td>
</tr>
<tr>
<td>77</td>
<td>COBOL Example of CEEDCOD</td>
<td>253</td>
</tr>
<tr>
<td>78</td>
<td>PL/I Example of CEEDCOD</td>
<td>255</td>
</tr>
<tr>
<td>79</td>
<td>C/C++ Example of CEEDSHP</td>
<td>258</td>
</tr>
<tr>
<td>80</td>
<td>COBOL Example of CEEDSHP</td>
<td>259</td>
</tr>
<tr>
<td>81</td>
<td>PL/I Example of CEEDSHP</td>
<td>260</td>
</tr>
<tr>
<td>82</td>
<td>C/C++ Example of CEEDYWK</td>
<td>263</td>
</tr>
<tr>
<td>83</td>
<td>COBOL Example of CEEDYWK</td>
<td>264</td>
</tr>
<tr>
<td>84</td>
<td>PL/I Example of CEEDYWK</td>
<td>266</td>
</tr>
<tr>
<td>85</td>
<td>C/C++ Example of CEEFMDA</td>
<td>268</td>
</tr>
<tr>
<td>86</td>
<td>COBOL Example of CEEFMDA</td>
<td>269</td>
</tr>
<tr>
<td>87</td>
<td>PL/I Example of CEEFMDA</td>
<td>270</td>
</tr>
<tr>
<td>88</td>
<td>C/C++ Example of CEEFMDT</td>
<td>272</td>
</tr>
<tr>
<td>89</td>
<td>COBOL Example of CEEFMDT</td>
<td>273</td>
</tr>
<tr>
<td>90</td>
<td>PL/I Example of CEEFMDT</td>
<td>274</td>
</tr>
<tr>
<td>91</td>
<td>COBOL Example of CEEFMON</td>
<td>277</td>
</tr>
<tr>
<td>92</td>
<td>PL/I Example of CEEFMON</td>
<td>279</td>
</tr>
<tr>
<td>93</td>
<td>C/C++ Example of CEEFMOTM</td>
<td>281</td>
</tr>
<tr>
<td>94</td>
<td>COBOL Example of CEEFMOTM</td>
<td>282</td>
</tr>
<tr>
<td>95</td>
<td>PL/I Example of CEEFMOTM</td>
<td>283</td>
</tr>
<tr>
<td>96</td>
<td>C/C++ Example of CEEFRST</td>
<td>286</td>
</tr>
<tr>
<td>97</td>
<td>COBOL Example of CEEFRST</td>
<td>287</td>
</tr>
<tr>
<td>98</td>
<td>PL/I Example of CEEFRST</td>
<td>288</td>
</tr>
<tr>
<td>99</td>
<td>COBOL Example of CEEFTDS</td>
<td>293</td>
</tr>
<tr>
<td>100</td>
<td>PL/I Example of CEEFTDS</td>
<td>295</td>
</tr>
<tr>
<td>101</td>
<td>C/C++ Example of CEEGMT</td>
<td>298</td>
</tr>
<tr>
<td>102</td>
<td>COBOL Example of CEEGMT</td>
<td>299</td>
</tr>
<tr>
<td>103</td>
<td>PL/I Example of CEEGMT</td>
<td>300</td>
</tr>
<tr>
<td>104</td>
<td>C/C++ Example of CEEGMTTO</td>
<td>303</td>
</tr>
<tr>
<td>105</td>
<td>COBOL Example of CEEGMTTO</td>
<td>303</td>
</tr>
<tr>
<td>106</td>
<td>PL/I Example of CEEGMTTO</td>
<td>304</td>
</tr>
<tr>
<td>107</td>
<td>C/C++ Example of CEEGPID</td>
<td>306</td>
</tr>
<tr>
<td>1. How to Use z/OS Language Environment Publications</td>
<td>xviii</td>
<td></td>
</tr>
<tr>
<td>2. Run-Time Options Quick Reference - AMODE 31</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3. Run-Time Options Quick Reference - AMODE 64</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4. Syntax examples</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5. Condition Handling of 0Cx ABENDS.</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>6. Handling of Software Raised Conditions</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>7. TRAP Run-Time Option Settings</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>8. Bit Manipulation Routines.</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>9. Condition Handling Callable Services</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>10. Date and Time Callable Services</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>11. Dynamic Storage Callable Services</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>12. General Callable Services</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>13. Initialization and Termination Services</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>14. Locale Callable Services</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>15. Math Services</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>16. Message Handling Callable Services</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>17. National Language Support and National Language Architecture Callable Services</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>18. Files Used in C/C++, COBOL, and PL/I Examples</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>19. Imbedding Files in Your Routines</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>20. Data Type Definitions for C/C++</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>21. Data Type Definitions for COBOL</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>22. Data Type Definitions for PL/I</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>23. National Language Codes</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>25. HEAP Attributes Based on the Setting of the options Parameter</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>26. Sample Output of CEEDATE</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>27. Sample Output of CEEDATM</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>28. Defaults Currency and Picture Strings Based on COUNTRY Setting</td>
<td>515</td>
<td></td>
</tr>
<tr>
<td>29. Picture Character Terms Used in Picture Strings for Date and Time Services</td>
<td>519</td>
<td></td>
</tr>
<tr>
<td>30. Examples of Picture Strings Recognized by Date and Time Services</td>
<td>520</td>
<td></td>
</tr>
<tr>
<td>31. Japanese Eras Used by Date/Time Services When &lt;JJJJ&gt; Specified</td>
<td>520</td>
<td></td>
</tr>
</tbody>
</table>
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 C/C++
- OS/390® C/C++
- C/C++ Compiler for MVS/ESA™
- C/C++ Compiler for z/VM
- AD/Cycle® C/370™ Compiler
- VisualAge for Java, Enterprise Edition for OS/390
- Enterprise COBOL for z/OS
- COBOL for OS/390 & VM
- COBOL for MVS & VM (formerly COBOL/370)
- Enterprise PL/I for z/OS and OS/390
- VisualAge PL/I for OS/390
- PL/I for MVS & VM (formerly PL/I MVS™ & VM)
- VS FORTRAN and FORTRAN IV (in compatibility mode)

Restrictions: The following restrictions apply to z/OS.e:

- The following compilers are not licensed for use on z/OS.e:
  - COBOL
  - PL/I
  - FORTRAN

- The following subsystems are not licensed for use on z/OS.e:
  - CICS
  - IMS™

- Execution of applications written in the following languages is not functionally supported on z/OS.e:
  - COBOL (except for precompiled COBOL DB2® stored procedures and other precompiled COBOL applications using the Language Environment preinitialization interface
  - FORTRAN

- The following are not functional and/or not licensed for use on z/OS.e:
  - Language Environment Library Routine Retention (LRR)
  - Language Environment compatibility preinitialization for C and PL/I

Customers are not permitted to use lower levels of Language Environment on z/OS.e.

Language Environment supports, but is not required for, an interactive debug tool for debugging applications in your native z/OS environment. The IBM interactive Debug Tool is available with z/OS, or with the latest releases of the C/C++, COBOL, PL/I and VisualAge for Java compiler products.
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. The first usage of every term listed in the glossary is indicated by *italics*. You can find the definitions for these terms in the [z/OS Language Environment Concepts Guide](#).

### 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 “How to specify run-time options” on page 9.

<table>
<thead>
<tr>
<th>To ...</th>
<th>Use ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate Language Environment</td>
<td><strong>z/OS Language Environment Concepts Guide</strong></td>
</tr>
<tr>
<td></td>
<td><strong>z/OS Language Environment Run-Time Application Migration Guide</strong></td>
</tr>
<tr>
<td>Install Language Environment</td>
<td><strong>z/OS Program Directory</strong></td>
</tr>
<tr>
<td>Customize Language Environment</td>
<td><strong>z/OS Language Environment Customization</strong></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>
</table>
| Understand Language Environment program models and concepts | z/OS Language Environment Concepts Guide  
z/OS Language Environment Programming Guide  
z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode |
| Find syntax for Language Environment run-time options and callable services | z/OS Language Environment Programming Reference |
| Develop applications that run with Language Environment | z/OS Language Environment Programming Guide and your language programming guide |
| Debug applications that run with Language Environment, diagnose problems with Language Environment | z/OS Language Environment Debugging Guide |
| Get details on run-time messages | z/OS Language Environment Run-Time Messages |
| Develop interlanguage communication (ILC) applications | z/OS Language Environment Writing  
Interlanguage Communication Applications and your language programming guide |
| Migrate applications to Language Environment | z/OS Language Environment Run-Time  
Application Migration Guide and the migration guide for each Language Environment-enabled language |

Where to find more information

Please see [z/OS Information Roadmap](http://www.ibm.com/servers/resourcelink) for an overview of the documentation associated with z/OS, including the documentation available for z/OS Language Environment.

Accessing z/OS licensed documents on the Internet

z/OS™ licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

http://www.ibm.com/servers/resourcelink

Licensed documents are available only to customers with a z/OS license; access to these documents requires an IBM Resource Link user ID and password, and a key code. Based on which offering you chose (ServerPac, CBPDO, SystemPac), information concerning the key code is available in the Installation Guide that is delivered with z/OS and z/OS.e orders as follows:

- ServerPac [Installing Your Order](http://www.ibm.com/servers/resourcelink)
- CBPDO [Memo to Users Extension](http://www.ibm.com/servers/resourcelink)
- SystemPac [Installation Guide](http://www.ibm.com/servers/resourcelink)

To obtain your IBM Resource Link user ID and password, log on to:

http://www.ibm.com/servers/resourcelink

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.
Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either z/OS Licensed Product Library CD-ROM or IBM Resource Link to print licensed documents.

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from the following locations to find IBM message explanations for z/OS elements and features, z/VM®, VSE/ESA™, and Clusters for AIX® and Linux:

- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e systems to access IBM message explanations, using LookAt from a TSO/E command line (for example, TSO/E prompt, ISPF, or z/OS UNIX® System Services).
- Your Microsoft® Windows® workstation. You can install code to access IBM message explanations on the z/OS Collection (SK3T-4269), using LookAt from a Microsoft Windows command prompt (also known as the DOS command line).
- Your wireless handheld device. You can use the LookAt Mobile Edition with a handheld device that has wireless access and an Internet browser (for example, Internet Explorer for Pocket PCs, Blazer, or Eudora for Palm OS, or Opera for Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt Web site.

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from a disk on your z/OS Collection (SK3T-4269), or from the LookAt Web site (click Download, and select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS and z/OS.e, see the online document at:

/publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS

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

Summary of Changes
for SA22-7562-06
z/OS Version 1 Release 6

The book contains information previously presented in z/OS Language Environment Programming Reference, SA22-7562-04, which supports z/OS Version 1 Release 5. It also clarifies information presented in SA22-7562-05, which supported z/OS Version 1 Release 6.

New Information

- A reference table for AMODE 64 options has been added to Chapter 1, “Language Environment run-time option reference tables and general information,” on page 3.
- Run-time options supporting AMODE 64 applications have been added to Chapter 2, “Using the Language Environment run-time options,” on page 11. These are:
  - HEAP64
  - HEAPPOOLS64
  - IOHEAP64
  - LIBHEAP64
  - STACK64
  - THREADSTACK64
- Several run-time options in Chapter 2, “Using the Language Environment run-time options,” on page 11, include parameters that allow them to be used for AMODE 64 applications.
- A section describing storage statistics for AMODE 64 applications has been added to Appendix C, “Controlling storage allocation,” on page 521.

Changed Information

- The publication has been divided into 3 parts to help make the information easier to retrieve.
- The reference table for AMODE 31 options has been updated in Chapter 1, “Language Environment run-time option reference tables and general information,” on page 3.
- Overall descriptions of several run-time options have been updated.

Deleted Information

- Support for RTLS has been removed, along with references to the SCEERTLS data set.
- Examples of storage reports and options reports have been moved to z/OS Language Environment Debugging Guide to avoid redundancy.

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

As part of an ongoing effort, several of the run-time options in Chapter 2, “Using the Language Environment run-time options,” on page 11, have been updated to improve the consistency and retrievability of the information.
Summary of Changes
for SA22-7562-04
z/OS Version 1 Release 5


New Information
- The HEAPCHK run-time option was updated to include the pool call depth suboption. The call level suboption was renamed to call depth.
- The HEAPPOOLS run-time option was updated with a 64 bit maximum size.

Changed Information
- The ENVAR run-time option in Chapter 2, “Using the Language Environment run-time options,” on page 11 has been updated to include the putenv() function and to more fully explain how the option works.
- Updates were made to the ERRCOUNT run-time option.
- Updates were made to the AUTOCVT suboption of the FILETAG run-time option.
- Updates were made to the HEAP run-time option.
- The options report in “RPTOPTS” on page 60 has been updated.
- The storage reports in “RPTSTG” on page 61 have been updated.
- Updates were made to the TERMTTHDACT run-time option.
- Updates were made to the CEE3DMP callable service.
- Updates were made to the CEEMOUT callable service.

As part of an ongoing effort, several of the run-time options in Chapter 2, “Using the Language Environment run-time options,” on page 11 have been updated to improve the consistency and retrievability of the information.

Summary of Changes
for SA22-7562-03
z/OS Version 1 Release 4

The book contains information previously presented in z/OS Language Environment Programming Reference, SA22-7562-02, which supports z/OS Version 1 Release 3.

New Information
- Information has been added to indicate that this book supports z/OS.e.

Changed Information
- The bit manipulation routines have been added to Chapter 1, “Language Environment run-time option reference tables and general information,” on page 3.
- Updates were made to the FILETAG run-time option.
- The options report in “RPTOPTS” on page 60 has been updated.
- The storage reports in “RPTSTG” on page 61 have been updated.

Moved Information
- “How to read syntax diagrams” has been moved to Chapter 2, “Using the Language Environment run-time options,” on page 11.
• The bit manipulation routines have been moved out of “Math services” on page 475 into Chapter 5, “Bit manipulation routines,” on page 469.

Deleted Information
• The CBLOPTS run-time option has been removed. It remains in z/OS Language Environment Customization.
• The appendix on Managed System Infrastructure for Setup (msys for Setup) has been removed.
• The NONIPTSTACK | NONONIPTSTACK run-time option has been removed because it is no longer supported. It was replaced by the THREADSTACK option.

Summary of Changes
for SA22-7562-02
z/OS Version 1 Release 3


New Information
• Added information about the vendor heap manager (VHM) to the HEAPPOOLS and RPTSTG run-time options.
• Added product number to the CEEGPID callable service.
• Added information to the RTEREUS run-time option.
• An appendix on Managed System Infrastructure for Setup (msys for Setup) has been added.
• An appendix with z/OS product accessibility information has been added.

You may notice changes in the style and structure of some content in this book — for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our books.

Summary of Changes
for SA22-7562-01
z/OS Version 1 Release 2

The book contains information previously presented in z/OS Language Environment Programming Reference, SA22-7562-00, which supports z/OS Version 1 Release 1.

New Information
• A change of the output parameter CEE_Version_ID for CEEGPID as a four-byte hex number in the form PPVVRM. For New Product Language Environment, the product number is 4.
• A new suboption has been added to the TERMTHDACT that will cause Language Environment to write the CEEDUMP on the CICS dump data set via an interface provided by CICS.
• A new FILETAG run-time option has been added.

Changed Information
• Changes to the run-time option defaults were made in Chapter 1, “Language Environment run-time option reference tables and general information,” on page 3 and Chapter 2, “Using the Language Environment run-time options,” on page 11.
This book contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.
Summary of Changes
for SA22-7562-00
z/OS Version 1 Release 1

This book contains information also presented in *OS/390 Language Environment for OS/390 & VM Programming Reference*. Some additional function is provided by APAR:

- **APAR PQ38324** — A new GENOPTS option has been added to CEE3DMP. This new option generates a run-time options report and places it at the end of the enclave information whenever the TRACE, UATRACE, DUMP and UADUMP options are invoked.

- **APAR PQ39927** and **PQ39636**— A new sub-option `reg_stor_amount` has been added to the TERMTHDACT run-time option. This new sub-option allows the user to control the amount of storage to be dumped when a CEEDUMP is produced. The amount can range from 0 (no storage dumped) to 256 bytes. The number specified will be rounded up to the nearest multiple of 32.
Part 1. 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 tables for AMODE 31 run-time options

The following tables are a quick reference of the Language Environment run-time options for AMODE 31 applications.

<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>11</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>12</td>
</tr>
<tr>
<td>AIXBLD</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>
<td>13</td>
</tr>
<tr>
<td>NOAIXBLD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL31</td>
<td>Indicates whether an application does or does not run entirely in AMODE(31).</td>
<td>14</td>
</tr>
<tr>
<td>ANYHEAP</td>
<td>Controls allocation of library heap storage not restricted to below the 16M line.</td>
<td>16</td>
</tr>
<tr>
<td>ARGPARSE</td>
<td>Specifies whether arguments on the command line are to be parsed in the usual C format.</td>
<td>17</td>
</tr>
<tr>
<td>AUTOTASK</td>
<td>Specifies whether Fortran Multitasking Facility is to be used by your program and the number of tasks that are allowed to be active.</td>
<td>18</td>
</tr>
<tr>
<td>BELOWHEAP</td>
<td>Controls allocation of library heap storage below the 16M line.</td>
<td>19</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>20</td>
</tr>
<tr>
<td>CBLPSHPOP</td>
<td>Controls whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL subprogram is called.</td>
<td>20</td>
</tr>
<tr>
<td>CBLQDA</td>
<td>Controls COBOL QSAM dynamic allocation.</td>
<td>21</td>
</tr>
<tr>
<td>CHECK</td>
<td>Indicates whether “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>DEPTHCONDLMT</td>
<td>Limits the extent to which conditions can be nested.</td>
<td>24</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Activates the COBOL batch debugging features specified by the “debugging lines” or the USE FOR DEBUGGING declarative.</td>
<td>24</td>
</tr>
<tr>
<td>NODEBUG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENV</td>
<td>Specifies the operating system that your C application runs under.</td>
<td>26</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>ENVAR</td>
<td>Sets the initial values for the environment variables.</td>
<td>26</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>28</td>
</tr>
<tr>
<td>ERRUNIT</td>
<td>Identifies the unit number to which run-time error information is to be directed.</td>
<td>29</td>
</tr>
<tr>
<td>EXECOPS</td>
<td>Specifies whether run-time options can be specified on the command line.</td>
<td>29</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>30</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>31</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>33</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>37</td>
</tr>
<tr>
<td>HEAPPOOLS</td>
<td>Specifies that (Quick) heap storage manager can be used.</td>
<td>38</td>
</tr>
<tr>
<td>INFOMSGFILTER</td>
<td>Allows the user to eliminate unwanted informational messages.</td>
<td>41</td>
</tr>
<tr>
<td>INQPCOPN</td>
<td>NOINQPCOPN</td>
<td>INQPCOPN controls whether the OPENED specifier on an INQUIRE by unit statement can be used to determine whether a preconnected unit has had any I/O statements directed to it.</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>Causes attentions recognized by the host operating system to be recognized by Language Environment.</td>
<td>43</td>
</tr>
<tr>
<td>LIBSTACK</td>
<td>Controls the allocation of the thread's library stack storage.</td>
<td>46</td>
</tr>
<tr>
<td>MSGFILE</td>
<td>Specifies the ddname of the run-time diagnostics file.</td>
<td>47</td>
</tr>
<tr>
<td>MSGQ</td>
<td>Specifies the number of ISI blocks allocated on a per-thread basis during execution.</td>
<td>50</td>
</tr>
<tr>
<td>NATLANG</td>
<td>Specifies the national language to use for the run-time environment.</td>
<td>51</td>
</tr>
<tr>
<td>OCSTATUS</td>
<td>NOOCSTATUS</td>
<td>Controls whether the OPEN and CLOSE status specifiers are verified.</td>
</tr>
<tr>
<td>PC</td>
<td>Specifies that Fortran static common blocks are not shared among load modules.</td>
<td>53</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>54</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>55</td>
</tr>
<tr>
<td>POSIX</td>
<td>Specifies whether the enclave can run with the POSIX semantics.</td>
<td>56</td>
</tr>
<tr>
<td>PROFILE</td>
<td>Controls the use of an optional profiler which collects performance data for the running application.</td>
<td>57</td>
</tr>
<tr>
<td>PRTUNIT</td>
<td>Identifies the unit number used for PRINT and WRITE statements that do not specify a unit number.</td>
<td>57</td>
</tr>
<tr>
<td>PUNUNIT</td>
<td>Identifies the unit number used for PUNCH statements that do not specify a unit number.</td>
<td>58</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>RDRUNIT</td>
<td>Identifies the unit number used for READ statements that do not specify a unit number.</td>
<td>58</td>
</tr>
<tr>
<td>REDIR</td>
<td>Specifies whether redirections for stdin, stderr, and stdout are allowed from the command line.</td>
<td>59</td>
</tr>
<tr>
<td>RECPAD</td>
<td>Specifies whether a formatted input record is padded with blanks.</td>
<td>59</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>50</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>51</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 whether 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>55</td>
</tr>
<tr>
<td>STORAGE</td>
<td>Controls the contents of storage that is allocated and freed.</td>
<td>59</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>71</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>79</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>81</td>
</tr>
<tr>
<td>TRACE</td>
<td>Determines whether Language Environment run-time library tracing is active.</td>
<td>84</td>
</tr>
<tr>
<td>TRAP</td>
<td>Specifies how Language Environment routines handle abends and program interrupts.</td>
<td>86</td>
</tr>
<tr>
<td>UPSI</td>
<td>Sets the eight UPSI switches on or off.</td>
<td>88</td>
</tr>
<tr>
<td>USRHDLR</td>
<td>Registers user condition handlers.</td>
<td>89</td>
</tr>
<tr>
<td>VCTRSAVE</td>
<td>Specifies whether any language in an application uses the vector facility when user-written condition handlers are called.</td>
<td>90</td>
</tr>
<tr>
<td>XPLINK</td>
<td>Controls the initialization of the XPLINK environment.</td>
<td>91</td>
</tr>
<tr>
<td>XUFLOW</td>
<td>Specifies whether an exponent underflow causes a program interrupt.</td>
<td>93</td>
</tr>
</tbody>
</table>

Quick reference tables for AMODE 64 run-time options

The following tables are a quick reference of the Language Environment run-time options for AMODE 64 applications.

<table>
<thead>
<tr>
<th>Run-Time Options</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGPARSE</td>
<td>Specifies whether arguments on the command line are to be parsed in the usual C format.</td>
<td>17</td>
</tr>
<tr>
<td>ENVAR</td>
<td>Sets the initial values for the environment variables.</td>
<td>26</td>
</tr>
</tbody>
</table>
### Table 3. Run-Time Options Quick Reference - AMODE 64 (continued)

<table>
<thead>
<tr>
<th>Run-Time Options</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECOPS</td>
<td>Specifies whether run-time options can be specified on the command line.</td>
<td>29</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>31</td>
</tr>
<tr>
<td>HEAP64</td>
<td>Controls allocation of the user heap.</td>
<td>35</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>37</td>
</tr>
<tr>
<td>FHEAPPOOLS64</td>
<td>Specifies that (Quick) heap storage manager can be used.</td>
<td>40</td>
</tr>
<tr>
<td>INFOMSGFILTER</td>
<td>Allows the user to eliminate unwanted informational messages.</td>
<td>41</td>
</tr>
<tr>
<td>IOHEAP64</td>
<td>Controls allocation of I/O heap storage.</td>
<td>44</td>
</tr>
<tr>
<td>LIBHEAP64</td>
<td>Controls allocation of library heap storage.</td>
<td>45</td>
</tr>
<tr>
<td>NATLANG</td>
<td>Specifies the national language to use for the run-time environment.</td>
<td>51</td>
</tr>
<tr>
<td>POSIX</td>
<td>Specifies whether the enclave can run with the POSIX semantics.</td>
<td>56</td>
</tr>
<tr>
<td>PROFILE</td>
<td>Controls the use of an optional profiler which collects performance data for the running application.</td>
<td>57</td>
</tr>
<tr>
<td>REDIR</td>
<td>Specifies whether redirections for stdin, stderr, and stdout are allowed from the command line.</td>
<td>59</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>80</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>81</td>
</tr>
<tr>
<td>STACK64</td>
<td>Controls the allocation and management of stack storage.</td>
<td>88</td>
</tr>
<tr>
<td>STORAGE</td>
<td>Controls the contents of storage that is allocated and freed.</td>
<td>89</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>71</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>THREADSTACK64</td>
<td>Controls the allocation of thread-level stack storage for both the upward and downward-growing stack.</td>
<td>83</td>
</tr>
<tr>
<td>TRACE</td>
<td>Determines whether Language Environment run-time library tracing is active.</td>
<td>84</td>
</tr>
<tr>
<td>TRAP</td>
<td>Specifies how Language Environment routines handle abends and program interrupts.</td>
<td>86</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.

Symbols

The following symbols may be displayed in syntax diagrams:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SM590000/SM590000</td>
<td>Indicates the beginning of the syntax diagram.</td>
</tr>
<tr>
<td>──/SM590000</td>
<td>Indicates that the syntax diagram is continued to the next line.</td>
</tr>
<tr>
<td>──/SM590000</td>
<td>Indicates that the syntax is continued from the previous line.</td>
</tr>
<tr>
<td>──/SM590000/SM630000</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 4. Syntax examples

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required item.</td>
<td>/SM590000/SM590000 KEYWORD required_item</td>
</tr>
<tr>
<td>Required item.</td>
<td>/SM590000/SM630000 KEYWORD required_choice1, required_choice2</td>
</tr>
<tr>
<td>Optional item.</td>
<td>/SM590000/SM590000 KEYWORD optional_item</td>
</tr>
<tr>
<td>Optional item.</td>
<td>/SM590000/SM590000 KEYWORD optional_choice1, optional_choice2</td>
</tr>
<tr>
<td>Default.</td>
<td>/SM590000/SM590000 KEYWORD default_choice1, optional_choice2, optional_choice3</td>
</tr>
<tr>
<td>Variable.</td>
<td>/SM590000/SM590000 KEYWORD variable</td>
</tr>
<tr>
<td>Repeatable item.</td>
<td>/SM590000/SM590000 KEYWORD repeatable_item</td>
</tr>
<tr>
<td>Repeatable item.</td>
<td>/SM590000/SM590000 KEYWORD repeatable_item</td>
</tr>
<tr>
<td>Fragment.</td>
<td>/SM590000/SM590000 KEYWORD fragment</td>
</tr>
</tbody>
</table>

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.

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.

Repeatable item.

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

A character within the arrow means you must separate repeated items with that character.

An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.

Fragment.

The ─ 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.
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 ignore these options. No messages are issued. For example, when the STACK option is sent across from AMODE 31 to AMODE 64, it is ignored. This is because the AMODE 64 application uses the STACK64 option. No attempt to convert the AMODE 31 option to the new AMODE 64 option is performed.
Chapter 2. Using the Language Environment 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 the one specified.

Restriction: 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

Non-CICS default: ABPERC(NONE)

CICS default: ABPERC is ignored under CICS.
ABPERC

Syntax

```
ABPerce (NONE abcode)
```

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

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

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

**z/OS UNIX consideration**

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

**Usage notes**

- Language Environment ignores ABPERC(S0Cx). 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.

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

**CICS default:** 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](https://www.ibm.com/support/docview.zhtml?com-doc-id=197012) for more information.
- To gather information about the unhandled condition, see “TERMTHDACT” on page 71.

**For more information**


---

**AIXBLD (COBOL Only)**

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Alternate Index Build</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AIXBLD</td>
</tr>
</tbody>
</table>

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

AIXBLD conforms to the ANSI 1985 COBOL standard.

**Non-CICS default:** NOAIXBLD

**CICS default:** AIXBLD is ignored under CICS.
NOAIXBLD/NOAIX

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

AIXBLD/AIX

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

z/OS UNIX consideration

- If you also specify the MSGFILE 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 47 for information about the MSGFILE run-time option.

ALL31

ALL31 specifies whether an application can run entirely in AMODE 31 or whether 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.

Non-CICS default: ALL31(ON)

CICS default: ALL31(ON)
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.

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 default 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)
- 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.
ALL31

- 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.
- **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 65 for information about the STACK run-time option.
- See **XPLINK** on page 91 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.

**Non-CICS default:** ANYHEAP(16K,8K,ANYWHERE,FREE)

**CICS default:** ANYHEAP(4K,4080,ANYWHERE,FREE)

**Syntax**

```
ANYheap( init_size, incr_size, ANYWHERE, FREE, KEEP)
```

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

- **incr_size**
  Determines the minimum size of any subsequent increment to the
anywhere heap. 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.

**ANYWHERE|ANY**

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

**BELOW**

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

**FREE**

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

**KEEP**

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

**CICS 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 61 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>
</table>

Chapter 2. Using the Language Environment run-time options 17
ARGPARSE | NOARGPARSE

ARGPARSE specifies whether 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.

Non-CICS default: ARGPARSE

CICS default: ARGPARSE is ignored under CICS.

AMODE 64 default: ARGPARSE

Syntax

ARGPARSE

NOARGPARSE

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.

Usage notes

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

For more information

- See “REDIR | NOREDIR (C Only)” on page 59 for a description of REDIR.

AUTOTASK | NOAUTOTASK (Fortran only)

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

Non-CICS default: NOAUTOTASK

CICS default: AUTOTASK is ignored under CICS.

Syntax

NOAUTOTASK

AUTOTASK (—loadmod,—numtasks)

NOAUTOTASK
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).

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

Syntax
\[
\text{BELOWHEAP}(\text{init\_size}, \text{incr\_size}, \text{FREE/KEEP})
\]

\text{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.

\text{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.

\text{FREE}
Specifies that storage allocated to BELOWHEAP increments is released when the last of the storage is freed.

\text{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.
BELOWHEAP

- When ALL31(OFF) is in effect, Language Environment allocates thread-specific control blocks from the below heap.

Usage notes

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

Performance consideration

You can improve performance with the BELOWHEAP run-time option by specifying values that minimize the number of times the operating system allocates storage. See ["RPTSTG" on page 61] 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

COBOL OPTionS

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

Since you can specify this option only in CEEUOPT, CEEROPT or CEEDEOPT at initialization, see ["z/OS Language Environment Customization"] for details.

CBLPSHPOP (COBOL only)

Derivation

COBOL PUSH POP

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

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.

Non-CICS default: CBLPSHPOP is ignored.

CICS default: CBLPSHPOP(ON)
Syntax

```
CBLPshpop (ON OFF)
```

**ON**  Automatically issues the following when a COBOL subroutine is called:
- An EXEC CICS PUSH HANDLE command as part of the routine initialization.
- An EXEC CICS POP HANDLE command as part of the routine termination.

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

Performance consideration

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

For more information

- For more information about CEEUOPT, see [z/OS Language Environment Customization](https://www.ibm.com/support/knowledgecenter/STXKQF_2.4.0/com.ibm.zos.v2r4.languagenv_customization.doc).  
- To determine where to locate information about CICS commands, see *CICS Master Index*.

**CBLQDA (COBOL only)**

**Derivation**

COBOL QSAM Dynamic Allocation

CBLQDA controls COBOL QSAM dynamic allocation on an OPEN statement.

**Non-CICS default:** CBLQDA(OFF)

**CICS default:** CBLQDA is ignored under CICS.

Syntax

```
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.
CBLQDA

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).

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.

Non-CICS default: CHECK(ON)

CICS default: CHECK(ON)

Syntax

```
CHECK (ON OFF)
```

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

Usage notes

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

Performance consideration

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

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 CEE3CTY callable service.
Non-CICS default: COUNTRY(US)

CICS default: COUNTRY(US)

Syntax

```
->COUNTRY-(country_code)->
```


\( \text{country}_\text{code} \)

A 2-character code that indicates to Language Environment the country on which to base the default settings. Table 28 on page 515 contains a list of valid country codes.

Usage notes

- If you specify an unsupported \( \text{country}_\text{code} \), Language Environment accepts the value and issues an informational message.

  If an unsupported \( \text{country}_\text{code} \) is specified on the COUNTRY run-time option and one of the services listed in National Language Support callable services on page 101 is called with the optional \( \text{country}_\text{code} \) 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 120.

- For more information about the RPTOPTS run-time option, see “RPTOPTS” on page 60.

- For more information about the RPTSTG run-time option, see “RPTSTG” on page 61.

- 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 443.

- For more information on setlocale(), see Z/OS C/C++ Programming Guide.
**DEBUG (COBOL only)**

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

Non-CICS default: NODEBUG

CICS default: NODEBUG

```
Syntax

NODEBUG

DEBUG
```

**NODEDEBUG**

Suppresses the COBOL batch debugging features.

**DEBUG**

Activates the COBOL batch debugging features.

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

**Usage notes**

- For information on specifying this option in CEEDOPT (CEECOPT), CEEROPT or CEEUOPT, see [z/OS Language Environment Customization](https://www.ibm.com/support/knowledgecenter/POWER9/pag101). Use DEBUG and NODEBUG only on the command line.

**Performance consideration**

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

**For more information**


**DEPTHCONDLMT**

```
Derivation

DEPTH of nested CONDITION LIMIT
```

DEPTHCONDLMT specifies the extent to which conditions can be nested. [Figure 1 on page 25](https://www.ibm.com/support/knowledgecenter/POWER9/pag101) illustrates the effect of DEPTHCONDLMT(3) on condition handling. The initial condition and two nested conditions are handled in this example. The third nested condition is not handled.
Non-CICS default: DEPTHCONDLMT(10)

CICS default: DEPTHCONDLMT(10)

Syntax

```
DEPTHCONDLMT(limit)
```

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

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

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

**z/OS UNIX 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*. 
ENV

ENV (C only)

Derivation

ENVironment

ENV specifies the operating system for your C application. C++ users can get the same behavior by using the z/OS 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.

Non-CICS default: 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 default: ENV is ignored under CICS.

Syntax

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

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

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

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.

**Non-CICS default: ENVAR("")**

**CICS default: ENVAR("")**

**AMODE 64 default: ENVAR("")**

### Syntax

```
ENVAR(name=value)
```

*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

- **Restrictions:**
  - 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 60.
- For more information on `getenv()`, `putenv()`, `setenv()`, `clearenv()`, `system()`, and the exec and spawn family of functions, see "z/OS 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.

Non-CICS default: ERRCOUNT(0)

CICS default: ERRCOUNT(0)

Syntax

```
ERRcount(number)
```

number

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

z/OS UNIX 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**.

For more information

• For more information in condition handling, see the [z/OS Language Environment Programming Guide](#).

---

**ERRUNIT (Fortran only)**

### Derivation

<table>
<thead>
<tr>
<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.

**Non-CICS default:** **ERRUNIT(6)**

**CICS default:** **ERRUNIT** is ignored under CICS.

### Syntax

```
ERRUnit
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 whether you can enter run-time options on the command line.
**EXECOPS | NOEXECOPS**

**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.

**Non-CICS default:** EXECOPS

**CICS default:** This option is ignored under CICS.

**AMODE 64 default:** EXECOPS

**Syntax**

```
/SM590000/SM590000 EXECOPS
/SM590000/SM630000 NOEXECOPS
```

**EXECOPS**

Specifies that you can enter run-time options on the command line. 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's parsed, see [z/OS Language Environment Programming Guide](https://www.ibm.com/support/docview.wss?uid=swg27050164).

**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.

**Non-CICS default:** FILEHIST

**CICS default:** FILEHIST is ignored under CICS.

**Syntax**

```
/SM590000/SM590000 Filehist
/SM590000/SM630000 NOFilehist
```

For more information about the format of the argument string and how it's parsed, see [z/OS Language Environment Programming Guide](https://www.ibm.com/support/docview.wss?uid=swg27050164).
FILEHIST
Causes the history of a file to be used in determining its existence. It checks to see whether:
• The file was ever internally opened (in which case it exists)
• The file was deleted by a CLOSE statement (in which case it does not exist).

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)

Derivation
FILE TAGging

FILETAG controls automatic text conversion and automatic file tagging for HFS files.

Guideline: 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 C/C++ Programming Guide for more information.

Non-CICS default: FILETAG(NOAUTOCVT,NOAUTOTAG)

CICS default: FILETAG is ignored under CICS.

AMODE 64 default: FILETAG(NOAUTOCVT,NOAUTOTAG)

Syntax

NOAUTOCVT
Disables automatic text conversion as described below.
AUTOCVT
Enables automatic text conversion for untagged HFS files opened using fopen() or freopen() in text mode. 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 C/C++ Programming Guide for more information on the _BPXK_CCSIDS environment variable.
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).

This suboption does not affect untagged HFS files that are automatically tagged by the AUTOTAG suboption. An HFS 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 BPXPRM.** member on your system.

For more information on the _BPXK_AUTOCVT environment variable, see [z/OS 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**

- **Guidelines:**
  - 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 whether 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 whether the file exists or if it cannot determine the size.

FLOW (COBOL only)

FLOW controls the FLOW output produced by OS/VS COBOL programs.

Restriction: You cannot set FLOW during Language Environment installation.

Non-CICS default: NOFLOW

CICS default: NOFLOW

Syntax

```
/SM590000/SM630000
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

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

CICS default: HEAP(4K,4080,ANYWHERE,KEEP,4K,4080)
**Syntax**

```
HEAP(
  init_size, incr_size, ANYWHERE,
  KEEP, FREE, initsz24, incrsz24)
```

**init_size**
Determine the initial allocation of heap storage. This value can be specified as \( n, nK, \) or \( nM \) 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**
Determine the minimum size of any subsequent increment to the user heap storage. This value can be specified as \( n, nK, \) or \( nM \) bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

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

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

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

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

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

initsz24
Determines the minimum initial size of user heap storage that is obtained below the 16M line for AMODE 24 applications that specify ANYWHERE in the HEAP run-time option. 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.

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 \( n, nK, \) or \( nM \) 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 61 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 215.  
- For more information about the CEEGTST callable service, see "CEEGTST—Get heap storage" on page 320.  
- For more information about the RPTSTG run-time option, see "RPTSTG" on page 61.

**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.
AMODE 64 default: HEAP64(1M,1M,KEEP,32K,32K,KEEP,4K,4K,FREE)

### Syntax

```
HEAP64([init64], [incr64], ..., KEEP, ..., FREE, [init31], [incr31])
```

- **init64**: Determines the initial allocation of 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 user heap storage obtained above the 2G bar. Specify this value as \(n\)M bytes of storage. If a value less than 0 is specified, the default is used.
- **KEEP**: Specifies that an increment to user heap storage is not released when the last of the storage within that increment is freed.
- **FREE**: Specifies that an increment to user heap storage is released when the last of the storage within that increment is freed.
- **init31**: Determines the minimum initial size of user heap storage that is obtained above the 16M line and below the 2G bar. This value can be specified as \(n\), \(n\)K, or \(n\)M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.
- **incr31**: Determines the minimum size of any subsequent increment to user heap storage that is obtained above the 16M line and below the 2G bar. This value can be specified as \(n\), \(n\)K, or \(n\)M number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.
- **init24**: Determines the minimum initial size of user heap storage that is obtained below the 16M line. This value can be specified as \(n\), \(n\)K, or \(n\)M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.
- **incr24**: Determines the minimum size of any subsequent increment to user heap storage that is obtained below the 16M line. This value can be specified as \(n\), \(n\)K, or \(n\)M number of bytes. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

### z/OS UNIX 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 61 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 Language Environment heap storage, see the IBM z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode.
- For more information about the RPTSTG run-time option, see “RPTSTG” on page 61.

HEAPCHK

**Derivation**

| HEAP storage CHecKing |

HEAPCHK performs diagnostic tests against the user heap.

**Non-CICS default:** `HEAPCHK(OFF,1,0,0,0)`

**CICS default:** `HEAPCHK(OFF,1,0,0,0)`

**AMODE 64 default:** `HEAPCHK(OFF,1,0,0,0)`

**Syntax**

```
HEAPChk (OFF, ON, frequency, delay, call depth, pool call depth)
```

**OFF**
Indicates that user heap checking is inactive.

**ON**
Indicates that user heap checking is active.

**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 nth call to a Language Environment heap storage management service. A value of zero results in no check for damage to the user heap.

**delay**
A value that causes a delay before user heap is checked for damage, and is specified in n, 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 nth call to a Language Environment heap storage management service.

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

**Guideline:** 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 displayed in the traceback for the trace of each heap pools GET or FREE for the heap storage diagnostics report. A value of zero is the default that turns heap pools GET or FREE storage diagnostics reporting off.

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

### 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, ensure that a SYSMDUMP DD is available. No CEEDUMP is produced.
- 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 and a number \( n \) greater than zero for call depth. For example (HEAPCHK(ON,0,0,10)).
- Under normal termination conditions, if the call depth is greater than zero and the frequency is set to zero, the heap storage diagnostics report is written to the CEEDUMP report, independent of the TERMTHDACT setting.
- You can also generate a heap storage diagnostics report by calling CEE3DMP with the BLOCKS option.
- **Guideline:** Since HEAPCHK does not validate individual cells within a cell pool, you should specify HEAPPOOLS(OFF) when running HEAPCHK to diagnose storage overlays in the heap.

### 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](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.5/com.ibm.zos.zos.debuggd.doc/TOC.html).

## HEAPPOOLS (C/C++ only)

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

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 multi-threaded 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.

Non-CICS default:
HEAPPools(Off,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,10,0,10,0,10,0,10,0,10)

CICS default:
HEAPPools(Off,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,10,0,10,0,10,0,10,0,10)

Syntax

```
HEAPPools((OFF|ON),<cell size><percentage>)
```

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

**ON**  
Specifies that the heap pools algorithm is used.

**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).

**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 HEAPPOLS 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 HEAPPOLS 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...
HEAPPOOLS

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.

- The HEAPCHK run-time option does not validate individual heap pool cells.
- Use of a vendor heap manager (VHM) overrides the use of the HEAPPOOLS run-time option.

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.

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

Derivation

<table>
<thead>
<tr>
<th>HEAP storage POOLS</th>
</tr>
</thead>
</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 multi-threaded C/C++ applications with a high frequency of calls to malloc(), calloc(), realloc(), free(), and operators new and delete. When active, heap pools virtually eliminates contention for user heap storage.

AMODE 64 default: HEAPPOOLS64(OFF,8,4000,32,2000,128,700,256,350,1024,100,2048,50,3072,50,4096,50,8192,25,16384,10,32768,5,65536,5)

Syntax

```
HEAPPOOLS64(OFF,ON,cs1,cc1,cs2,cc2,...,cs12,cc12)
```

- **OFF** Specifies that the AMODE 64 heappools algorithm is not used.
- **ON** Specifies that the AMODE 64 heappools algorithm is used.
- **cs1 — cs12** 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). The minimum cell count is 4.
- **cc1 — cc12** 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.
- The HEAPCHK run-time option does not validate individual heap pool cells.

**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.

**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/SSEPV7_11.3.0/com.ibm.zos.v1r12.doc/LEPG/index.html).

---

**INFOMSGFILTER**

<table>
<thead>
<tr>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFOrmational MeSsaGe FILTER</td>
</tr>
</tbody>
</table>

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.

**Non-CICS default:** INFOMSGFILTER(OFF,,)

**CICS default:** INFOMSGFILTER(OFF,,)

**AMODE 64 default:** INFOMSGFILTER(OFF,,
INFOMSGFILTER

Syntax

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

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

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

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

**Background**  
Selecting this keyword causes message filtering to be turned on for the following environments:
- MVS Batch

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

**Non-CICS default: INQPCOPN**

**CICS default: INQPCOPN is ignored under CICS.**

Syntax

```
INQpcopn
```

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.

**Non-CICS default:** INTERRUPT(OFF)

**CICS default:** INTERRUPT is ignored under CICS.

### Syntax

```plaintext
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.

**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 77 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.

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

Syntax

```
IOHEAP64 (init64, incr64, KEEP, FREE),

init31, incr31, KEEP, FREE,

init24, incr24, 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 \(n\), \(n\)K, or \(n\)M number of bytes. If 0 is specified, the initial storage is obtained on the first use and is based on the increment size. The amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

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

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

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

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

### 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 61.

### LIBHEAP64 (AMODE 64 only)

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

**AMODE 64 default: LIBHEAP64(1M,1M,FREE,16K,8K,FREE,8K,4K,FREE)**

#### Syntax

```plaintext
LIBHEAP64( init64 , incr64 , KEEP , FREE )

init31 , incr31 , KEEP , FREE , init24 , incr24

KEEP

FREE
```

- **init64** Determines the initial allocation of library heap storage obtained above the 2G bar. Specify this value as nM bytes of storage. If a value of 0 or less is specified, the default is used.
- **incr64** Determines the minimum size of any subsequent increment to the library heap storage obtained above the 2G bar. Specify this value as nM bytes of storage. If a value less than 0 is specified, the default is used.
- **KEEP** Specifies that an increment to library heap storage is not released when the last of the storage within that increment is freed.
- **FREE** Specifies that an increment to library heap storage is released when the last of the storage within that increment is freed.
- **init31** Determines the minimum initial size of library heap storage that is obtained above the 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 library 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 library 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 library 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**

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

**For more information**

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

**LIBSTACK**

**Derivation**

**Libr ary STACK storage**

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

**Non-CICS default:** LIBSTACK(4K,4K,FREE)

**CICS default:** LIBSTACK(32,4080,FREE)

**Syntax**

```
LIBStack(init_size, incr_size), FREE
```

**init_size**

Determines the minimum size of the initial library stack storage. This value can be specified as \( n \), \( nK \), or \( nM \) bytes of storage. Language Environment allocates the storage rounded up to the nearest multiple of 8 bytes.

**incr_size**

Determines the minimum size of any subsequent increment to the library stack. This value can be specified as \( n \), \( nK \), or \( nM \) bytes of storage. The
actual amount of allocated storage is the larger of 2 values— incr_size or
the requested size—rounded up to the nearest multiple of 8 bytes.

If you specify 0 as incr_size, Language Environment gets only the amount
of storage needed at the time of the request, rounded up to the nearest
multiple of 8 bytes.

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

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

CICS 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 notes

• 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 61] 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 61] 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

<table>
<thead>
<tr>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeSsaGe FILE</td>
</tr>
</tbody>
</table>

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

Non-CICS default: MSGFILE(SYSOUT,FBA,121,0,NOENQ)
MSGFILE

CICS default: MSGFILE is ignored under CICS.

Syntax

```
MSGFILE(\[ddname\], recfm, lrecl, blksize, \[NOENQ\], \[ENQ\])
```

- `ddname` The ddname of the message file where Language Environment directs the information listed above.
- `recfm` The record format (RECFM) for the message file. `recfm` is used when this information is not available either in a data set definition or in the label of an existing data set. The following record formats are acceptable: F, FA, FB, FBA, FBS, FBSA, U, UA, V, VA, VB, and VBA.
- `lrecl` The record length (LRECL) for the message file. `lrecl` is used when this information is not available either in a data set definition or in the label of an existing data set. `lrecl` is expressed as bytes of storage. The `lrecl` cannot exceed `blksize`. For variable-length record formats, the `lrecl` is limited to `blksize` minus 4.
- `blksize` The block size (BLKSIZE) for the message file. `blksize` is used when this information is not available either in a data set definition or in the label of an existing data set. `blksize` is expressed as bytes of storage and cannot exceed 32760.

**NOENQ**

Specifies that no serialization is performed on `ddname`.

**ENQ**

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

CICS consideration

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

z/OS UNIX 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 whether 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

• **Guideline:** 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 whether 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 *rectfm* specifies unblocked fixed-length format records (F or FA) or undefined-format records (U or UA), *blksize* is the same as *lrecl*.
  – If *rectfm* specifies unblocked variable-length format records (V or VA), *blksize* is *lrecl* plus 4.
  – If *rectfm* 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 *rectfm* specifies blocked fixed-length format records (FB, FBA, FBS, or FBSA) for a terminal, *blksize* is the same as *lrecl*.
  – If *rectfm* 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 *rectfm*, *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 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).

• 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.
MSGFILE

For more information

- For more information about the RPTOPTS and RPTSTG run-time options, see "RPTOPTS" on page 60 and "RPTSTG" on page 61.
- For more information about the CEEMSG and CEEMOUT callable services, see "CEEMSG—Get, format, and dispatch a message" on page 394 and "CEEMOUT—Dispatch a message" on page 371.
- For details on how HLL compiler options affect messages, see information on HLL I/O statements and message handling in the 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 394.
- For more information about perror() and stderr see C message output information in the z/OS Language Environment Programming Guide.
- For more information about the CESE transient data queue, see the z/OS Language Environment Programming Guide.

MSGQ

Derivation
MeSsaGe_Queue

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

Non-CICS default: MSGQ(15)

CICS default: 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 "CEECEMI—Store and load message insert data" on page 208.
NATLANG

Derivation

NATIONAL LANGUAGE

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.

Non-CICS default: NATLANG(ENU)

CICS default: NATLANG(ENU)

AMODE 64 default: NATLANG(ENU)

Syntax

```
>>NATlang((ENU)(UEN)(JPN))
```

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

UEN A 3-character ID specifying uppercase U.S. English.
- Message text consists of SBCS characters and includes only uppercase letters.

JPN A 3-character ID specifying Japanese.
- Message text can contain a mixture of SBCS and DBCS characters.

Usage notes

- **Restriction:** CEE3LNG and CEESETL are not available to AMODE 64 applications.
- You can use the CEE3LNG callable service to set the national language.
- If you specify a national language that is not available on your system, Language Environment uses the IBM-supplied default ENU (mixed-case U.S. English).
- Language Environment writes storage reports, option reports, and dump output in mixed-case U.S. English only.
- 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 163.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 443.
- For more information on setlocale(), see z/OS C/C++ Programming Guide.

OCSTATUS (Fortran only)

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

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

Non-CICS default: OCSTATUS

CICS default: OCSTATUS is ignored under CICS.

Syntax

```plaintext
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.

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

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

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

---

**PC (Fortran only)**

**Derivation**

Private Common blocks

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

Non-CICS default: NOPC

CICS default: 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.

**Guideline:** 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.

**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.

**Non-CICS default: PLIST(HOST)**

**CICS default:** PLIST is ignored under CICS.

**Syntax**

```
PLIST({ HOST | CICS | CMS | IMS | MVS | OS | TSO })
```

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

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

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

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

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

OS

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

TSO

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

Usage notes

- The behavior of C applications with PLIST(HOST) in effect is the same for C++.
- When using the pre-Language Environment-conforming C interface for 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 C++ compiler option PLIST(OS).
- z/OS UNIX consideration—The PLIST option applies only to the main routine of the initial thread.

PLITASKCOUNT (PL/I only)

**Derivation**

PL/I TASK COUNTer

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

**Non-CICS default: PLITASKCOUNT(20)**

**CICS default:** PLITASKCOUNT is ignored under CICS.

**Syntax**

```
>>> PLITaskcount(tasks)
```

**tasks**

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

**Usage notes**

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

**POSIX**

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

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

- **Non-CICS default: POSIX(OFF)**
- **CICS default: POSIX is ignored under CICS**
- **AMODE 64 default: POSIX(OFF)**

**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 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.
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 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.

Non-CICS default: PROFILE(OFF,"")

CICS default: PROFILE(OFF,"")

AMODE 64 default: PROFILE(OFF,"")

Syntax

PROFILE (OFF, ON, string)

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

For more information

• For a detailed description of the setup and use of the IBM z/OS C/C++
  PROFILER, see z/OS UNIX System Services User’s Guide

PRTUNIT (Fortran only)

Derivation

PRinT UNIT

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

Non-CICS default: PRTUNIT(6)

CICS default: 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.

Non-CICS default: PUNUNIT(7)

CICS default: PUNUNIT is ignored under CICS.

Syntax

```
   PUNunit(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

```
ReaDeR UNIT
```

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

Non-CICS default: RDRUNIT(5)

CICS default: RDRUNIT is ignored under CICS.
**RECPAD (Fortran only)**

**Syntax**

```plaintext
RECPAD [number]
```

*number*

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

**Usage notes**

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

**REDIR | NOREDIR (C Only)**

**Syntax**

```plaintext
REDIR | NOREDIR
```

**Derivation**

**REDIREction**

REDIR specifies whether 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.

**Non-CICS default:** REDIR

**CICS default:** REDIR is ignored under CICS.

**AMODE 64 default:** REDIR
REDIR | NOREDIR

**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 17 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.

Non-CICS default: RPTOPTS(OFF)

CICS default: RPTOPTS(OFF)

AMODE 64 default: RPTOPTS(OFF)

**Syntax**

```
RPTOPTS(ON OFF)
```

**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.
- RPTOPTS does 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.

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 47 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

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

You can use the storage report information to help you set the ANYHEAP, BELOWHEAP, HEAP, HEAP64, HEAPPOOLS, HEAPPOOLS64, IOHEAP64, LIBHEAP64, LIBSTACK, STACK, STACK64, THREADHEAP, THREADSTACK, and THREADSTACK64 run-time options for the best storage tuning.

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

Non-CICS default: RPTSTG(OFF)

CICS default: RPTSTG(OFF)

AMODE 64 default: RPTSTG(OFF)
RPTSTG

**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.
- RPTOPTS does not generate the options report if your application ends abnormally.
- 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.
- RPTSTG does not generate a storage report if your application terminates abnormally.
- 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](https://www.ibm.com/support/docview.ws/docview/110573).
- For more information about the MSGFILE run-time option, see “MSGFILE” on page 47.
- For an example and complete description of the storage report, see [z/OS Language Environment Debugging Guide](https://www.ibm.com/support/docview.ws/docview/110545).
RTEREUS (COBOL only)

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

Non-CICS default: NORTEREUS

CICS default: RTEREUS is ignored under CICS.

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.
RTEREUS

- 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.
- Use RTEREUS and NORTEREUS only on the command line.
- 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.
- See z/OS Language Environment Programming Guide for more information about preinitialization.

SIMVRD (COBOL only)

**Derivation**

SIMulate Variable length Relative organization Data sets

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

**Non-CICS default:** NOSIMVRD

**CICS default:** SIMVRD is ignored under CICS.

**Syntax**

```
NOSIMVRD
```

NOSIMVRD

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

For more information


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.

Non-CICS default: STACK(128K,128K,ANYWHERE,KEEP,512K,128K)

CICS default: STACK(4K,4080,ANYWHERE,KEEP,4K,4080)

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.
STACK

**Example:** In this 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 I FREE apply to any thread created using `pthread_create`. Language Environment gets the initial stack size from the threads attribute object specified in the `pthread_create`
function. The default size to be set in the thread’s attribute object is obtained from the STACK 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.

- 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 61 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 14 for more information about the ALL31 run-time option.

- See “RPTSTG” on page 61 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 91 for more information about the XPLINK run-time option.
STACK64

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.

AMODE 64 default: STACK64(1M,1M,128M)

Syntax

```
STACK64(initial, increment, maximum)
```

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

- **increment**  Determines the minimum size of any subsequent increment to the downward-growing stack area. This value is specified as \(n\)M bytes of storage. The actual amount of allocated storage is the larger of two values—increment or the requested size—rounded up to the nearest 1MB.

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

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

- **maximum**  Specifies the maximum stack size. This value is specified as \(n\)M bytes of storage.

Usage notes

- The 1MB guard page is not included in any of the sizes

Performance consideration

To improve performance, use the storage report numbers generated by the RPTSTG run-time option as an aid in setting the initial and increment sizes for STACK64.

For more information

- See "RPTSTG on page 61" 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.
STORAGE controls the initial content of storage when allocated and freed. It also controls the amount of storage that is reserved for the out-of-storage condition. If you specify one of the parameters in the STORAGE run-time option, all allocated storage processed by that parameter is initialized to the specified value. Otherwise, it is left uninitialized.

You can use the STORAGE option to identify uninitialized application variables, or prevent the accidental use of previously freed storage. STORAGE is also useful in data security. For example, storage containing sensitive data can be cleared when it is freed.

Non-CICS default: STORAGE(NONE,NONE,NONE,0K)

CICS default: STORAGE(NONE,NONE,NONE,0K)

AMODE 64 default: STORAGE(NONE,NONE,NONE,)

Syntax

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

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

- A single character enclosed in 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.

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 ‘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’.
- **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 CEEOPD 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 LE condition handling and messaging internal routines as well as the user condition handler. When using the reserve stack in a multi-threaded 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.

Usage notes

- `heap_alloc_value`, `heap_free_value`, and `dsa_alloc_value` can all be enclosed in quotes. To initialize heap storage to the EBCDIC equivalent of a single quote, double it within the string delimited by single quotes or surround it with a pair of double quotes. Both of the following are correct ways to specify a single quote:

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

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

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

- COBOL consideration—If using WSCLEAR in VS COBOL II, STORAGE(00,NONE,NONE,8K) is recommended.

Performance consideration

Using STORAGE to control initial values can increase program run-time. If you specify a `dsa_alloc_value`, performance is likely to be poor. Therefore, use the `dsa_alloc_value` option only for debugging, not to initialize automatic variables or data structures.

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

TERMTHDACT

<table>
<thead>
<tr>
<th>Derivation</th>
<th>TERMinating THreaD ACTions</th>
</tr>
</thead>
</table>

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 NOVARIABLES NOFILES STACKFRAME(ALL) PAGESIZE(60) FNAME(CEEDUMP) GENOPTS

The following options are passed for DUMP and UADUMP:
- THREAD(ALL) NOENTRY TRACEBACK FILES VARIABLES BLOCK STORAGE STACKFRAME(ALL) PAGESIZE(60) FNAME(CEEDUMP) CONDITION GENOPTS
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.

Non-CICS default: TERMTHDACT(TRACE,CESE,96)

CICS default: TERMTHDACT(TRACE,CESE,96)

AMODE 64 default: TERMTHDACT(TRACE,,96)

Syntax

```
TERMTHDACT(TRACE, QUIET, MSG, DUMP, UATRACE, UADUMP, UAONLY, UAIMM, CESE, CICSDDS, reg_stor_amount)
```

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE</td>
<td>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.</td>
</tr>
<tr>
<td>QUIET</td>
<td>Specifies that Language Environment does not generate a message when a thread terminates due to an unhandled condition of severity 2 or greater.</td>
</tr>
<tr>
<td>MSG</td>
<td>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.</td>
</tr>
<tr>
<td>DUMP</td>
<td>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.</td>
</tr>
<tr>
<td>UATRACE</td>
<td>Specifies that when a thread terminates due to an unhandled condition of severity 2 or greater, Language Environment generates a message indicating the cause of the termination, a trace of the active routines on the activation stack, and a U4039 system dump of the user address space. Under CICS, you will get a CICS transaction dump. Under non-CICS, if the appropriate DD statement is used, you will get a system dump of your user address space.</td>
</tr>
</tbody>
</table>
TERMTHDACT

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

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

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

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

CESE
Specifies that Language Environment dump output will be written to the CESE QUEUE as it has always been.

Restriction: 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.

Restriction: 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 in a non-64-bit environment. The default amount is 0 in a 64–bit environment.

CICS consideration

- All TERMTHDACT output is written to the data queue based on the setting of CESE or CICSDDS.

Table 5 on page 74 for help in understanding 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 queue.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td>• Traceback written to CESE queue.</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>• Message written to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td>• Traceback written to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• CEEDUMP to CESE queue.</td>
<td>• CEEDUMP to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• ASRA or user ABEND issued.</td>
<td>• ASRA or user ABEND issued.</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 included in CICS transaction dump for this ABEND.</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>• 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>• Message written to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• Traceback written to CESE queue.</td>
<td>• Traceback written to CESE queue.</td>
</tr>
<tr>
<td></td>
<td>• CEEDUMP written to CESE queue.</td>
<td>• CEEDUMP 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>• ASRA or user ABEND issued.</td>
<td>• ASRA or user ABEND issued.</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>
</table>
| DUMP    | • Message written to CESE queue or MSGFILE.  
          • Traceback written to CESE queue.  
          • CEE_DUMP written to CESE queue.  
          • U4038 abend issued.  | • Invalid sub-option combination. Not supported. |
| UATRACE | • Message written to CESE queue or MSGFILE.  
          • Traceback written to CESE queue.  
          • U4039 transaction dump in CICS dump data set.  
          • U4038 abend issued.  | • Message written to CESE queue or MSGFILE.  
          • Traceback written to CESE queue.  
          • U4039 transaction dump in CICS dump data set.  
          • U4038 abend issued.  |
| UADUMP  | • Message written to CESE queue or MSGFILE.  
          • Traceback written to CESE queue.  
          • CEE_DUMP written to CESE queue.  
          • U4039 transaction dump in CICS dump data set.  
          • U4038 abend issued.  | • Invalid sub-option combination. Not supported. |
| UAONLY  | • U4039 transaction dump in CICS dump data set.  
          • U4038 abend issued.  | • No changes in behavior for CICSDDS. |
| UAIMM   | • U4039 transaction dump in CICS dump data set.  
          • U4038 abend issued.  | • Incorrect sub-option combination. Not supported. |

**Note:**

- CICS is told about software raised error conditions for DUMP and TRACE.
- When assembling a CEECOPT, CEEROPT, or CEEUOPT, the CICSDDS option cannot be issued with DUMP or UADUMP. This results in a RC=8, CEEXOPT 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.  
- Language Environment requests a CICS transaction dump via the U4039 abend.
- See [z/OS Language Environment Run-Time Messages](https://www.ibm.com/support/knowledgecenter/SSEPK2_2.2.0/com.ibm.zos.v2r2.constants.doc/ceexoptt000.html) for more complete details regarding the U4039 abend.
- 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 and the following message occurs in the CESE queue:
  
  CEE3627I The following messages pertain to the programmer default run-time options.  
  CEE3775W A conflict was detected between the TERMTHDACT suboptions CICSDDS and DUMP.  
  The TERMTHDACT level setting has been set to TRACE.
and the traceback is written to the CICS transaction dump data set.

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

  Format of the environment variable is:

  _CEE_DMPTARG=value

  where value is a null-terminated character string defined as one of the following values:

  SYSOUT(x)—where x 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 an z/OS UNIX shell, you could issue the export command and specify the following run-options, for example:

  export _CEE_DMPTARG=SYSOUT(A)

  76   z/OS V1R6.0 Language Environment Programming Reference
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 84 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 127.
- See z/OS Language Environment Programming Guide for more information about the TERMTHDACT run-time option and condition message.
- For More Information about CESE, see z/OS Language Environment Programming Guide.

TEST | NOTEST

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

Non-CICS default: NOTEST(ALL, *, PROMPT, INSPPREF)

CICS default: NOTEST(ALL, *, PROMPT, INSPPREF)

AMODE 64 default: NOTEST(ALL, *, PROMPT, INSPPREF)

Syntax

```
/SM590000/SM590000
/SM590000
/SM590000
/SM590000

NOTest TEST

ALL

ERROR

NONE

commands_file

PROMPT

NOPROMPT

*command

INSPPREF

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
TEST I NOTEST

- 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 I 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.

INSPREF
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.
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 User’s Guide and Reference* for details and for examples of the TEST run-time option as it relates to Debug Tool.

#### THREADHEAP

<table>
<thead>
<tr>
<th><strong>Derivation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>THREAD level HEAP storage</td>
</tr>
</tbody>
</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.

**Non-CICS default:** THREADHEAP(4K,4K,ANY,KEEP)

**CICS default:** THREADHEAP is ignored under CICS.
Syntax

```
THREADHeap( init_size, incr_size, ANYWHERE, KEEP, FREE )
```

*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/I MTF considerations—The thread-level heap is allocated only in applications that use the PL/I MTF. For PL/I MTF applications, controlled and based variables specified in subtasks are located in the thread-level heap.
If the main program is running in 24-bit addressing mode and THREADHEAP(,,ANY,) is in effect, heap storage is allocated below the 16M line. The only case in which storage is allocated above the line is when all of the following conditions exist:
- The user routine requesting the storage is running in 31-bit addressing mode.
- HEAP(,,ANY,,,) is in effect.
- The main routine is running in 31-bit addressing mode.

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

**THREADSTACK**

**Derivation**

THREAD level STACK storage

THREADSTACK controls the allocation of the thread’s stack storage for both the upward and downward-growing stacks, except the initial thread in a multi-threaded application.

If the thread attribute object does not provide an explicit stack size, then the allocation values can be inherited from the STACK option or specified explicitly on the THREADSTACK option.

**Non-CICS default:** THREADSTACK(OFF,4K,4K,ANYWHERE,KEEP,128K,128K)

**CICS default:** THREADSTACK is ignored under CICS.

**Syntax**

```
THREADStack
  (OFF, ON, usinit_size, usincr_size, ANYWHERE, KEEP, ANY, 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 \( n \), \( nK \), or \( nM \) bytes of storage. The actual amount of allocated storage is rounded up to the nearest multiple of 8 bytes.

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

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.

- 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 NONONIPTSTACK options.

## THREADSTACK64 (AMODE 64 only)

### Derivation

THREAD level STACK storage

THREADSTACK64 controls the allocation of the thread's stack storage for the downward-growing stack, except the initial thread in a multi-threaded application.

**AMODE 64 default:** THREADSTACK64(OFF,1M,1M,128M)

### Syntax

```
THREADSTACK64 (OFF, 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.
THREADSTACK64

ON

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

initial

Determines the size of the initial stack segment. The storage is contiguous. This value is specified as \( n \) \( M \) bytes of storage.

increment

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

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

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

maximum

Specifies the maximum stack size. This value is specified as \( n \) \( M \) bytes of storage.

Usage notes

- The 1MB guard page is not included in any of the sizes.

For more information

- For more information about using the storage reports generated by the RPTSTG run-time option to tune the stacks for AMODE 64 applications, see [z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode](#).

TRACE

TRACE controls 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 whether TRACE is active and the DUMP suboption is specified.

Non-CICS default: TRACE(OFF,4K,DUMP,LE=0)

CICS default: TRACE(OFF,4K,DUMP,LE=0)

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

Syntax

```
TRACE(ON,,table_size,,DUMP,,LE=0,,OFF,,NODUMP,,LE=1,,LE=2,,LE=3)
```

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

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

**table_size**
Determines the size of the tracing table as specified in bytes ($nK$ or $nM$).
The upper limit is 16M - 1 (1666777215 bytes).

**Restriction:** This option is ignored in a 64-bit environment and the size is set to 1M.

**DUMP**
Requests that a Language Environment-formatted dump (containing the trace table) be taken at program termination regardless of the setting of the TERMTHDACT run-time option.

**NODUMP**
Requests that a Language Environment-formatted dump not be taken at program termination.

**LE=0**
Specifies that no trace events be recorded.

**LE=1**
Specifies that entry to and exit from Language Environment member libraries be recorded (such as, in the case of C, entry and exit of the `printf()` library function).

**LE=2**
Specifies that mutex init/destroy and locks/unlocks from Language Environment member libraries be recorded.

**LE=3**
Activates both the entry/exit trace and the mutex trace.

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 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.

TRAP(ON) must be in effect for the ABTERMENC run-time option to have effect.

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

Table 7. TRAP Run-Time Option Settings

<table>
<thead>
<tr>
<th>If...</th>
<th>then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>a single 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>NOSTAE are specified,</td>
</tr>
<tr>
<td>multiple instances of SPIE</td>
<td>NOSPIE are specified,</td>
</tr>
<tr>
<td>an options string has TRAP(ON) or TRAP(OFF) together with SPIE</td>
<td>NOSPIE, and/or STAE</td>
</tr>
</tbody>
</table>

CEESGL is unaffected by this option.

Non-CICS default: TRAP(ON,SPIE)

CICS default: TRAP(ON,SPIE)

AMODE 64 default: TRAP(ON,SPIE)

Syntax

```
 TRAP(ON, SPIE)
```

ON Fully enables the Language Environment condition handler.

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

**SPIE**
SPIE specifies that Language Environment issue an ESPIE macro to handle program interrupts. The SPIE 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 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 below, 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.
- Running with TRAP(OFF) (for exception diagnosis purposes) can cause many side effects, because Language Environment uses condition handling internally and requires TRAP(ON). When you run with TRAP(OFF), you can get side effects even if you do not encounter a software-raised condition, program check, or abend. If you do encounter a program check or an abend with TRAP(OFF) in effect, the following side effects can occur:
  - 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 COMPUTE 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_CODES list in CEEBXITA, Language Environment percolates the abend. Normal Language Environment condition handling is never invoked to handle these abends. This feature is useful when you do not want Language Environment condition handling to intervene for certain abends or when you want to prevent invocation of the abnormal termination exit for certain abends, such as when IMS issues a user ABEND code 777.
- When TRAP(ON,NOSPIE) is specified in a non-CICS environment, Language Environment will handle program interrupts and abends via an ESTAE. This feature is useful when you do not want Language Environment to issue an ESPIE macro.
- When TRAP(OFF), (TRAP(OFF,SPIE) or TRAP(OFF,NOSPIE) is specified and there is a program interrupt, the user exit for termination is not driven.
- z/OS UNIX consideration—The TRAP option applies to the entire enclave and all threads within.

For more information
- See "ABTERMENC" on page 12 for more information about the ABTERMENC run-time option.
- See "CEESGL—Signal a condition" on page 449 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 325.
- 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>
</tr>
</thead>
<tbody>
<tr>
<td>User Programmable Status Indicator</td>
</tr>
</tbody>
</table>

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

Non-CICS default: UPSI(00000000)

CICS default: UPSI(00000000)

Syntax

```plaintext
$UPSI (nnnnnnnn)
```
nnnnnnnn

\( n \) represents one UPSI switch between 0 and 7, the leftmost \( n \) representing the first switch. Each \( n \) can either be 0 (off) or 1 (on).

**CICS consideration**

- The default under CICS is \( \text{UPSI}=(00000000) \).

**Usage notes**

- Specify this option in CEEDOPT(CEECOPT), CEEROPT or CEEUOPT with a string of eight binary flags; for example: \( \text{UPSI}(00000000) \). 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 Programming Guide](#) or [COBOL for OS/390 & VM Programming Guide](#).

**USRHDLR | NOUSRHDLR**

**Derivation**

\[ \text{USRHDLR} \]

\[ \text{NOUSRHDLR} \]

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

**Non-CICS default: NOUSRHDLR**

**CICS default: NOUSRHDLR()**

**Syntax**

\[ \text{NOUSRHDLR} \]

\[ \text{USRHDLR} \]

\[ \text{lmname} \]

\[ \text{lmname2} \]

**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.

\[ \text{lmname} \]

- 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.

\[ \text{lmname2} \]

- The name of a load module (or an alias name of a load module) that contains
the user condition handler that is to be registered to get control after the enablement phase and before any other user condition handler.

**CICS consideration**

- When specifying USRHDLR under CICS, `lmname` and `lmname2` must be defined in the CICS PPT.

**Usage notes**

- **Restriction:** If user condition handler `lmname` is in effect, it is unsupported to resume execution in the program in which the condition occurs. This includes calls in the condition handler to CEEMRCR and CEEMRCE. This restriction does not apply to user condition handler `lmname2`.
- 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 `lmname` is invoked for conditions that are still unhandled after being presented to condition handlers for the main program.
- The user condition handler `lmname2` is invoked for each condition after the condition completes the enablement phase but before any other registered user condition handler is 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 `lmname` and `lmname2` occurs only when that user condition handler needs to be invoked the first time.
- If the load of either `lmname` or `lmname2` fails, an error message is issued.
- Modules specified as user handlers that contain one of the fenced languages for the environment will cause a condition to be signaled on the z/OS.e operating system. The condition token is CEE38V.

**For more information**

- For information on registering a user condition handler and its interfaces, see "CEEHDLR—Register User-written condition handler" on page 325.

---

**VCTRSAVE**

**Derivation**

| VeCToR environment to be SAVEd |

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

**Non-CICS default:** VCTRSAVE(OFF)

**CICS default:** VCTRSAVE is ignored under CICS.
Syntax

```
VCTRSAVE (OFF | ON)
```

**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.

**Usage notes**

- 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**

**Derivation**

```
eXtra Performance LINKage
```

The XPLINK run-time option controls the initialization of the XPLINK environment. XPLINK resources such as the downward-growing stack and the XPLINK 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).
Non-CICS default: XPLINK(OFF)

CICS default: XPLINK is ignored under CICS.

### Syntax

```
XPLINK(ON | OFF)
```

- **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.

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

### 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 degration 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 affect, 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 (ex. COBOL, 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>
</tr>
</thead>
<tbody>
<tr>
<td>eXponent Under FLOW</td>
</tr>
</tbody>
</table>

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

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

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

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

Non-CICS default: XUFLOW(AUTO)

CICS default: XUFLOW(AUTO)
Syntax

| AUTO | An exponent underflow causes or does not cause a program interrupt dynamically, based upon the HLLs that make up the application. Enablement is determined without user intervention. XUFLOW(AUTO) causes condition management to process underflows only in those applications where the semantics of the application languages require it. Normally, XUFLOW(AUTO) provides the best efficiency while meeting language semantics. |
| ON  | An exponent underflow causes a program interrupt. XUFLOW(ON) causes condition management to process underflows regardless of the mix of languages; therefore, this setting might be less efficient in applications that consist of languages not requiring underflows to be processed by condition management. |
| OFF | An exponent underflow does not cause a program interrupt; the hardware takes care of the underflow. When you set XUFLOW to OFF, the hardware processes exponent underflows. This is more efficient than condition handling to process the underflow. |

Usage notes

- PL/I consideration—When setting XUFLOW to OFF, be aware that the semantics of PL/I require the underflow to be signaled.
- 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
Chapter 3. Quick reference tables for Language Environment services

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

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 chapters in this book.

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>469</td>
</tr>
<tr>
<td>CEESISET</td>
<td>Bit set</td>
<td>469</td>
</tr>
<tr>
<td>CEESISHF</td>
<td>Bit shift</td>
<td>470</td>
</tr>
<tr>
<td>CEESITST</td>
<td>Bit test</td>
<td>470</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>116</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>148</td>
</tr>
<tr>
<td>CEE3GRO</td>
<td>Returns the offset within a failing routine of the most recent condition.</td>
<td>156</td>
</tr>
<tr>
<td>CEE3SPM</td>
<td>Queries and modifies the enablement of Language Environment hardware conditions.</td>
<td>189</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>197</td>
</tr>
<tr>
<td>CEEDCOD</td>
<td>Decodes an existing condition token.</td>
<td>249</td>
</tr>
<tr>
<td>CEEGQDT</td>
<td>Provides a mechanism by which application code can retrieve the q_data_token from the Instance Specific Information (ISI).</td>
<td>309</td>
</tr>
<tr>
<td>CEEHDLR</td>
<td>Registers a user condition handler for the current stack frame.</td>
<td>325</td>
</tr>
<tr>
<td>CEEHDLU</td>
<td>Unregisters a user-written condition handler for the current stack frame.</td>
<td>335</td>
</tr>
<tr>
<td>CEEITOK</td>
<td>Computes the initial condition token for the current condition information block.</td>
<td>348</td>
</tr>
<tr>
<td>CEEMRCE</td>
<td>Resumes execution of a user routine at the location established by CEE3SRP.</td>
<td>375</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>CEEMRCR</td>
<td>Moves the resume cursor to a position relative to the current position of the handle cursor.</td>
<td>382</td>
</tr>
<tr>
<td>CEENCOD</td>
<td>Dynamically constructs a condition token.</td>
<td>400</td>
</tr>
<tr>
<td>CEESGL</td>
<td>Raises, or signals, a condition to the condition manager.</td>
<td>449</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>203</td>
</tr>
<tr>
<td>CEEDATE</td>
<td>Converts a number representing a Lilian date to a date written in character format.</td>
<td>227</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>234</td>
</tr>
<tr>
<td>CEEDAYS</td>
<td>Converts a string representing a date to a Lilian format.</td>
<td>242</td>
</tr>
<tr>
<td>CEEDYWK</td>
<td>Calculates the day of the week on which a Lilian date falls.</td>
<td>261</td>
</tr>
<tr>
<td>CEEGMT</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>296</td>
</tr>
<tr>
<td>CEEGMTO</td>
<td>Computes values to the calling routine that represent the difference between the local system time and Greenwich Mean Time.</td>
<td>300</td>
</tr>
<tr>
<td>CEEISEC</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>341</td>
</tr>
<tr>
<td>CEELOCT</td>
<td>Gets the current local time in three formats.</td>
<td>360</td>
</tr>
<tr>
<td>CEEQCEN</td>
<td>Queries the century within which Language Environment assumes two-digit year values lie.</td>
<td>407</td>
</tr>
<tr>
<td>CEESCCEN</td>
<td>Sets the century in which Language Environment assumes two-digit year values lie.</td>
<td>421</td>
</tr>
<tr>
<td>CEESECI</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>430</td>
</tr>
<tr>
<td>CEESECS</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>435</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>CEE3RPH</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>186</td>
</tr>
<tr>
<td>CEECRHP</td>
<td>Creates additional heaps.</td>
<td>215</td>
</tr>
<tr>
<td>CEECZST</td>
<td>Changes the size of a previously allocated storage element, while preserving its contents.</td>
<td>220</td>
</tr>
<tr>
<td>CEEDSHP</td>
<td>Discards an entire heap that you created previously with a call to CEECRHP.</td>
<td>256</td>
</tr>
<tr>
<td>CEEFRST</td>
<td>Frees storage previously allocated by CEEGTST.</td>
<td>283</td>
</tr>
<tr>
<td>CEEGTST</td>
<td>Allocates storage from a heap whose ID you specify.</td>
<td>320</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>CEE3DMP</td>
<td>Generates a Language Environment formatted dump of the run-time environment and member language libraries.</td>
<td>127</td>
</tr>
<tr>
<td>CEE3PRM</td>
<td>Returns to the calling routine the argument string that was specified at program invocation.</td>
<td>183</td>
</tr>
<tr>
<td>CEE3USR</td>
<td>Sets or queries one of two 4-byte fields known as the user area fields.</td>
<td>198</td>
</tr>
<tr>
<td>CEEGPID</td>
<td>Retrieves the version of Language Environment currently in use.</td>
<td>304</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>418</td>
</tr>
<tr>
<td>CEETEST</td>
<td>Invokes a tool for debugging.</td>
<td>462</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>113</td>
</tr>
<tr>
<td>CEE3GRC</td>
<td>Gets the enclave return code.</td>
<td>135</td>
</tr>
<tr>
<td>CEE3SRC</td>
<td>Sets the enclave return code.</td>
<td>196</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>274</td>
</tr>
</tbody>
</table>
### Table 14. Locale Callable Services (continued)

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEEFTDS</td>
<td>Formats time and date into a character string.</td>
<td>289</td>
</tr>
<tr>
<td>CEELCNV</td>
<td>Queries locale numeric conventions.</td>
<td>354</td>
</tr>
<tr>
<td>CEEQDTC</td>
<td>Queries locale date and time conventions and returns the specified format information.</td>
<td>410</td>
</tr>
<tr>
<td>CEEQRYL</td>
<td>Queries the active locale environment.</td>
<td>416</td>
</tr>
<tr>
<td>CEESCOL</td>
<td>Compares the collation weights of two strings.</td>
<td>426</td>
</tr>
<tr>
<td>CEESETL</td>
<td>Sets the locale operating environment.</td>
<td>443</td>
</tr>
<tr>
<td>CEESTXF</td>
<td>Transforms string characters into collation weights.</td>
<td>455</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>475</td>
</tr>
<tr>
<td>CEESxACS</td>
<td>Arccosine</td>
<td>476</td>
</tr>
<tr>
<td>CEESxASN</td>
<td>Arcsine</td>
<td>477</td>
</tr>
<tr>
<td>CEESxATH</td>
<td>Hyperbolic arctangent</td>
<td>478</td>
</tr>
<tr>
<td>CEESxATN</td>
<td>Arctangent</td>
<td>479</td>
</tr>
<tr>
<td>CEESxAT2</td>
<td>Arctangent of two arguments</td>
<td>480</td>
</tr>
<tr>
<td>CEESxCJG</td>
<td>Conjugate complex</td>
<td>480</td>
</tr>
<tr>
<td>CEESxCOS</td>
<td>Cosine</td>
<td>481</td>
</tr>
<tr>
<td>CEESxCSH</td>
<td>Hyperbolic cosine</td>
<td>483</td>
</tr>
<tr>
<td>CEESxCTN</td>
<td>Cotangent</td>
<td>484</td>
</tr>
<tr>
<td>CEESxDIM</td>
<td>Positive difference</td>
<td>486</td>
</tr>
<tr>
<td>CEESxDVD</td>
<td>Division</td>
<td>487</td>
</tr>
<tr>
<td>CEESxERC</td>
<td>Error function complement</td>
<td>487</td>
</tr>
<tr>
<td>CEESxERF</td>
<td>Error function</td>
<td>488</td>
</tr>
<tr>
<td>CEESxEXP</td>
<td>Exponential (base e)</td>
<td>489</td>
</tr>
<tr>
<td>CEESxGMA</td>
<td>Gamma function</td>
<td>490</td>
</tr>
<tr>
<td>CEESxIMG</td>
<td>Imaginary part of complex</td>
<td>491</td>
</tr>
<tr>
<td>CEESxINT</td>
<td>Truncation</td>
<td>492</td>
</tr>
<tr>
<td>CEESxLGM</td>
<td>Log gamma function</td>
<td>493</td>
</tr>
<tr>
<td>CEESxLG1</td>
<td>Logarithm base 10</td>
<td>494</td>
</tr>
<tr>
<td>CEESxLG2</td>
<td>Logarithm base 2</td>
<td>495</td>
</tr>
<tr>
<td>CEESxLOG</td>
<td>Logarithm base e</td>
<td>496</td>
</tr>
<tr>
<td>CEESxMLT</td>
<td>Floating-point complex multiplication</td>
<td>497</td>
</tr>
<tr>
<td>CEESxMOD</td>
<td>Modular arithmetic</td>
<td>497</td>
</tr>
<tr>
<td>CEESxNIN</td>
<td>Nearest integer</td>
<td>499</td>
</tr>
<tr>
<td>CEESxNWN</td>
<td>Nearest whole number</td>
<td>499</td>
</tr>
<tr>
<td>CEESxSGN</td>
<td>Transfer of sign</td>
<td>500</td>
</tr>
</tbody>
</table>
### Table 15. Math Services (continued)

<table>
<thead>
<tr>
<th>Math Service</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESxSIN</td>
<td>Sine</td>
<td>501</td>
</tr>
<tr>
<td>CEESxSNH</td>
<td>Hyperbolic sine</td>
<td>503</td>
</tr>
<tr>
<td>CEESxSQRT</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>507</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>CEECMI</td>
<td>Stores message insert data.</td>
<td>208</td>
</tr>
<tr>
<td>CEEMGET</td>
<td>Retrieves a message corresponding to a condition token returned from a callable service.</td>
<td>364</td>
</tr>
<tr>
<td>CEEMOUT</td>
<td>Writes a specified message to the message string file.</td>
<td>371</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>394</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>120</td>
</tr>
<tr>
<td>CEE3LNG</td>
<td>Changes or queries the current national language.</td>
<td>163</td>
</tr>
<tr>
<td>CEE3MCS</td>
<td>Gets the default currency symbol for a country specified in country_code.</td>
<td>172</td>
</tr>
<tr>
<td>CEE3MDS</td>
<td>Gets the default decimal separator for the country specified in country_code.</td>
<td>175</td>
</tr>
<tr>
<td>CEE3MTS</td>
<td>Gets the default thousands separator for the country that you specify in country_code.</td>
<td>179</td>
</tr>
<tr>
<td>CEEFMDA</td>
<td>Gets the default date picture string for a country specified in country_code.</td>
<td>267</td>
</tr>
<tr>
<td>CEEFMDT</td>
<td>Gets the default date and time picture strings for the country specified in country_code.</td>
<td>270</td>
</tr>
<tr>
<td>CEEFMTM</td>
<td>Gets the default time picture string for the country specified in country_code.</td>
<td>280</td>
</tr>
</tbody>
</table>
Chapter 4. Language Environment callable services

Restriction: Language Environment callable 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 C/C++
- IBM OS/390 C/C++
- C/C++ for MVS/ESA
- IBM SAA AD/Cycle C/370
- Enterprise COBOL for z/OS and OS/390
- Enterprise PL/I for z/OS and OS/390
- IBM COBOL for OS/390 & VM
- IBM COBOL for MVS & VM
- IBM PL/I for MVS & VM
- VisualAge PL/I for OS/390

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:

- "General usage notes for callable services" on page 104
- "Invoking callable services" on page 105
- "Data type definitions" on page 110
- "Callable services" on page 113
NOTE
For a list 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

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

```call ceedate using x,y,z. **invalid**```

**Figure 2. An Invalid COBOL CALL that Omitsthe fc Parameter**

```dcl ceedate entry options(asm);
call ceedate(x,y,z); /* invalid */```

**Figure 3. An Invalid PL/I CALL that Omitsthe fc Parameter**

```call ceedate using x,y,z,omitted.
call ceedate using x,y,z,fc.```

**Figure 4. Valid COBOL CALLs that Use the Optional fc Parameter**
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.

The names and descriptions of the files imbedded in the callable service examples in this chapter are provided in Table 18.

Table 18. Files Used in C/C++, COBOL, and PL/I Examples

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEEEDCCT</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 used to explicitly omit the fc parm, for routines written in C/C++</td>
</tr>
<tr>
<td>CEEIGZCI</td>
<td>COBOL declarations for condition information block</td>
</tr>
<tr>
<td>CEEIGZCT</td>
<td>COBOL 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, and LC_TIME.</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>
</tbody>
</table>

Figure 5. Valid PL/I CALLs that Use the Optional fc Parameter
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>
<tbody>
<tr>
<td>CEEIGZDT</td>
<td>DTCONV structure needed for 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>PL/I declarations for condition information block</td>
</tr>
<tr>
<td>CEEIBMCT</td>
<td>PL/I declarations for Language Environment symbolic feedback codes</td>
</tr>
<tr>
<td>CEEIBMLC</td>
<td>Locale category constants LC_ALL, LC_COLLATE, LCCTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.</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 CEEQCNV calls</td>
</tr>
<tr>
<td>CEEIBMMDT</td>
<td>DTCONV structure needed for CEEQDTC calls</td>
</tr>
<tr>
<td>CEEFORCT</td>
<td>Fortran declarations for Language Environment symbolic feedback codes</td>
</tr>
</tbody>
</table>

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

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:

Table 19. Imbedding Files in Your Routines

<table>
<thead>
<tr>
<th>To imbed a file in a...</th>
<th>Use...</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ routine</td>
<td><code>#include</code> statement</td>
</tr>
<tr>
<td>COBOL program</td>
<td><code>COPY</code> statement</td>
</tr>
<tr>
<td>PL/I routine</td>
<td><code>%include</code> statement</td>
</tr>
</tbody>
</table>

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

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

```ceuXXX```(parm1, parm2, ...parmn, fc);
```ceuXXX```(parm1, parm2, ...parmn, NULL);
```

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

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

```
Syntax

```ceuXXX```(parm1, parm2, ...parmn, NULL);
```

See "Parameter list for invoking callable services" on page 109 for a description of this syntax.

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

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

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

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

_Figure 6. Sample Callable Services Invocation Syntax for C/C++_

To help in checking the success of your applications, Language Environment provides FBCHECK in the cceedcct.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:
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 109 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 below:

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

Figure 7. Sample Callable Services Invocation Syntax for COBOL

PL/I syntax

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

```plaintext
CALL "CEExxxx" USING parm1, parm2, ... parmn, OMITTED;
```

PL/I also allows you to omit arguments. In place of the argument, code an asterisk (*), as shown below:
Note that you cannot invoke callable services as function references.

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

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

If you code your own declarations for the Language Environment callable services, be sure to specify all required arguments in the CALL statement. Figure 3 on page 104 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 shown below:

```pli
%CINCLUDE CEEIBMAW;
CALL CEExxxx(parml, parm2, ... parmn, fc);
```

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

Parameter list for invoking callable services

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

**CEExxxx**

The name of the callable service.

By including a reference to a header file in your code, you can avoid declaring each callable service as an external entry. See “Header, copy, or include files” on page 105 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, Table 21 on page 111, and Table 22 on page 112 contain data type definitions and their descriptions for C/C++, COBOL, and PL/I, respectively.

### C/C++ data type definitions

The table below includes data type definitions and their descriptions 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 real and imaginary parts are each 4-byte single-precision floating-point numbers</td>
<td>Not available</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>Not available</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>Not available</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>
</tbody>
</table>
| VSTRING     | A halfword length-prefixed character string (for input); fixed-length 80-character string (for output) | struct _VSTRING(
  _INT2 short length;
  char string[1];
)string-in;
  char string_out[80]; |
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>FEED_BACK</td>
<td>A mapping of the condition token (fc)</td>
<td>Case 1: typedef struct { short tok_sev ; short tok_msgno ; int tok_case :2, tok_sever:3, tok_ctrl :3 ; char tok_facid[3] ; int tok_isi ; } _FEEDBACK ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Case 2: typedef struct { short tok_sev ; short tok_msgno ; int tok_class_code :2, tok_cause_code:3, tok_ctrl :3 ; char tok_facid[3] ; int tok_isi ; } _FEEDBACK ;</td>
</tr>
<tr>
<td>CEE_ENTRY</td>
<td>An HLL-dependent entry constant</td>
<td>FUNCTION POINTER</td>
</tr>
</tbody>
</table>

COBOL data type definitions

The table below includes data type definitions and their descriptions for COBOL:

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 9(4) USAGE IS BINARY</td>
</tr>
<tr>
<td>INT4</td>
<td>A 4-byte signed integer</td>
<td>PIC 9(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, 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>USAGE IS POINTER</td>
</tr>
<tr>
<td>CHARn</td>
<td>A string (character array) of length n</td>
<td>PIC X(n)</td>
</tr>
</tbody>
</table>
Table 21. Data Type Definitions for COBOL  (continued)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>COBOL</th>
</tr>
</thead>
</table>
| VSTRING     | A halfword length-prefixed character string (for input); fixed-length 80-character string (for output) | 01 STRING-IN.  
              |                                                                            | 02 LEN  PIC 9(4) USAGE IS BINARY.  
              |                                                                            | 02 TXT  PIC X(N).  
              |                                                                            | 01 STRING-OUT PIC X(80). |
| FEED_BACK   | A mapping of the condition token (fc)                                        | Case 1:  
              |                                                                            | 01 FC                                            |
|             |                                                                            | 02 SEV  PIC 9(4) USAGE IS BINARY.                                    |
|             |                                                                            | 02 MSGNO PIC 9(4) USAGE IS BINARY.                                    |
|             |                                                                            | 02 FLGS  PIC X(1).                                                     |
|             |                                                                            | 02 FACID  PIC X(3).                                                    |
|             |                                                                            | 02 ISI  PIC X(4).                                                      |
|             |                                                                            | Case 2:  
              |                                                                            | 01 FC                                            |
|             |                                                                            | 02 CLASS-CODE PIC 9(4) USAGE IS BINARY.                                |
|             |                                                                            | 02 CAUSE-CODE PIC 9(4) USAGE IS BINARY.                                |
|             |                                                                            | 02 FLGS  PIC X(1).                                                     |
|             |                                                                            | 02 FACID  PIC X(3).                                                    |
|             |                                                                            | 02 ISI  PIC X(4).                                                      |
| CEE_ENTRY   | An HLL-dependent entry constant                                              | USAGE IS PROCEDURE-POINTER                                           |

PL/I data type definitions

The table below 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>CHARn</td>
<td>A string (character array) of length n</td>
<td>CHAR(n)</td>
</tr>
</tbody>
</table>
Table 22. Data Type Definitions for PL/I (continued)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>PL/I</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>CEE_ENTRY</td>
<td>An HLL-dependent entry constant</td>
<td>ENTRY</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

```plaintext
>>>CEE3ABD—(—abcode—,—clean-up—)
```

*abcode*

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*

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Issue the abend without clean-up</td>
</tr>
<tr>
<td>1</td>
<td>Issue the abend with normal enclave termination processing</td>
</tr>
</tbody>
</table>

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 whether the application actually terminates with an abend.

**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**

**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.

**For more information**

- For more information about the ABPERC run-time option, see [“ABPERC” on page 11](#).
- For more information about the TRAP run-time option, see [“TRAP” on page 86](#).
- For more information about the TERMTHDACT run-time option, see [“TERMTHDACT” on page 71](#).

**Examples**

```c
/*Module/File Name: EDC3ABD */
#include <leawi.h>

int main(void) {
  _INT4 code, timing;
  code = 1234; /* Abend code to issue */
  timing = 0;
  CEE3ABD(&code,&timing);
}
```

*Figure 9. C/C++ Example of CEE3ABD*
CBL LIB,QUOTE

*Module/File Name: IGZT3ABD

*******************************************************************************
** CBL3ABD - Call CEE3ABD to terminate the **
** enclave with an abend **
*******************************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3ABD.

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

PROCEDURE DIVISION.
PARA-CBLMGET.

*******************************************************************************
** 3415 is the abend code to be issued,  **
** a timing of zero requests an abend  **
** without clean-up  **
*******************************************************************************

MOVE 3415 TO ABDCODE.
MOVE 0 TO TIMING.
CALL "CEE3ABD" USING ABDCODE , TIMING.
GOBACK.

Figure 10. COBOL Example of CEE3ABD

+PROCESS MACRO;
/*Module/File Name: IBM3ABD

*******************************************************************************
/** */
/** Function: CEE3ABD - terminate enclave with an */
/** abend */
/** */
/** In this example, CEE3ABD is called with a */
/** timing value of 1. This requests an abend that */
/** is deferred until clean-up takes place. */
/** */
*******************************************************************************

PL3ABD: PROC OPTIONS(MAIN);

  %INCLUDE CEEIBMMAW;
  %INCLUDE CEEIBMCT;

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

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

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

END PL3ABD;

Figure 11. PL/I Example of CEE3ABD
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*

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

/*Module/File Name: EDC3CIB */

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

#ifdef __cplusplus
extern "C"
#endif

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

#ifdef __cplusplus
}
#endif

int main(void) {

_FEEDBACK fc;
_ENTRY routine;
_INT4 token;
int x,y,z;

/* set the routine structure to point to the handler */
/* and use CEEHDLR to register the user handler */

token = 99;
routine.address = (POINTER)&handler;
routine.nesting = NULL;
CEEHDLR(&routine,&token,&fc);

Figure 12. C/C++ Example of EDC3CIB (Part 1 of 2)
/* verify that CEEHDLR was successful */
if (_FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
    exit(2999);
}

x = 5;
y = 0;
z = x / y;

///////////////////////////////////////////////////////////////////////////
/* handler is a user condition handler */
*******************************************************************************/

void handler(_FEEDBACK *fc , _INT4 *token , _INT4 *result ,
             _FEEDBACK *newFc) {

    _CEECIB *cib_ptr;
    _FEEDBACK cibfc;

    CEE3CIB(fc, &cib_ptr, &cibfc);

    /* verify that CEE3CIB was successful */
    if (_FBCHECK ( cibfc , CEE000 ) != 0 ) {
        printf("CEE3CIB failed with message number %d\n", cibfc.tok_msgno);
        exit(2999);
    }

    printf("%s \n",(*cib_ptr).cib_eye);
    printf("%d \n",cib_ptr->cib_cond.tok_msgno);
    printf("%s \n",cib_ptr->cib_cond.tok_facid);
    *result = 10;
    }

Figure 12. C/C++ Example of EDC3CIB (Part 2 of 2)
CBL LIB,QUOTE
*Module/File Name: IGZT3CIB
************************************************
**
** CBL3CIB - Register a condition handler **
** that will call CEE3CIB to get **
** a Condition Information Block **
** (CIB) for the condition. **
**
************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3CIB.

DATA DIVISION.
WORKING-STORAGE SECTION.
 01 ROUTINE PROCEDURE-POINTER.
 01 TOKEN PIC S9(9) BINARY.
 01 FC PIC X(12).
PROCEDURE DIVISION.
PARA-CBL3CIB.
  SET ROUTINE TO ENTRY "HANDLER".
  CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
  GOBACK.
END PROGRAM CBL3CIB.

CBL LIB,QUOTE
IDENTIFICATION DIVISION.
PROGRAM-ID. HANDLER.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 CIB-PTR POINTER.
 01 FC PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-HANDLER.
  CALL "CEE3CIB" USING CURCOND, TOKEN, FC.
  DISPLAY "In Handler".
  DISPLAY CIB-EYE.
  DISPLAY CIB-TOK-MSGNO.
  DISPLAY CIB-TOK-FACID.
  GOBACK.
END PROGRAM HANDLER.

Figure 13. COBOL Example of CEE3CIB
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.

---

**Figure 14. PL/I Example of CEE3CIB**

```pli
*PROCESS OPT(0), MACRO;
/* Module/File Name: IBM3CIB */
/***************************/
/**
/** Function: CEE3CIB - example of CEE3CIB 
/** invoked from PL/I ON-unit **/
/**
*******************************************************************************/
IBM3CIB: PROCEDURE OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;
%INCLUDE CEEIBMCI;

DECLARE CIB_PTR POINTER,
    FC,
    /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
    03 ISI /* Instance-Specific Information */
        REAL FIXED BINARY(31,0),
    divisor FIXED BINARY(31) INITIAL(0);

ON ZERODIVIDE BEGIN;
    CALL CEE3CIB(*, CIB_PTR, FC);
    IF FBCHECK(FC,CEE000) THEN DO;
        PUT SKIP LIST('Found '||CIB_PTR->CIB_EYE||'
    ' for message #'||CIB_PTR->CIB_TOK_MSGNO
    ' from '||CIB_PTR->CIB_TOK_FACID);
    END;
    ELSE DO;
        DISPLAY('CEE3CIB failed with msg'
                || FC.MsgNo);
    END;
    END /* ON ZeroDivide */;

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

END IBM3CIB;
```
CEE3CTY affects only the Language Environment NLS and date and time services, not the Language Environment locale callable services or C locale-sensitive functions.

The current default country setting, as well as previous default settings that have not been discarded, are stored on the stack on a LIFO (last in, first out) basis. The current default country setting is always on the top of the stack.

There are two methods of changing the default country setting with CEE3CTY:

- Specify the new default country setting and place it on top of the stack using a function value of 1 (SET). This discards the previous default setting.
- Specify the new default country setting and place it on top of the stack using a function value of 3 (PUSH). This pushes the previous default setting down on the stack so that you can later retrieve it by discarding the current setting.

To illustrate the second method, suppose you live in the United States and the code for the United States is specified as the default at installation. If you want to use the French defaults for a certain application, you can use CEE3CTY to PUSH France as the default country setting; then when you want the defaults for the United States, you can POP France from the top of the stack, making the United States the default setting.

### Syntax

```
CEE3CTY((function, country_code, fc))
```

#### function (input)
A fullword binary integer specifying the service to be performed.

The possible values for function are:

- **1—SET** Establishes the country_code parameter as the current country. The top of the stack is, in effect, replaced with country_code.
- **2—QUERY** Returns the current country code on the top of the stack to the calling routine. The current code is returned in the country_code parameter.
- **3—PUSH** Pushes the country_code parameter onto the top of the country code stack, making it the current country code. Previous country codes on the stack are retained on a LIFO basis, which makes it possible to return to a prior country code at a later time.
- **4—POP** Pops the current country code. The last country code that was affected by a PUSH now becomes the current country_code. On return to the calling routine, the country_code parameter contains the discarded country code. If the stack contains only one country code, the code cannot be popped because the stack would be empty after the call. Therefore, no action is taken and a feedback code indicating such is returned to the calling routine.
**CEE3CTY**

*country_code (input/output)*

A 2-character fixed-length string. *country_code* is not case-sensitive. Table 28 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 function is specified as:</th>
<th>then <em>country_code</em>:</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 28 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>
<tr>
<td>4</td>
<td>Returns the discarded 2-character country code. In this case, it is an output parameter.</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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
<td></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 CEE3CTY. The current country code was returned in the</td>
<td></td>
</tr>
<tr>
<td>CEE3C0</td>
<td>3</td>
<td>3456</td>
<td>The country code <em>country-code</em> for the PUSH or SET function for CEE3CTY was invalid. No operation was performed.</td>
<td></td>
</tr>
<tr>
<td>CEE3C1</td>
<td>3</td>
<td>3457</td>
<td>The function <em>function</em> specified for CEE3CTY was not recognized. No operation was performed.</td>
<td></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 60 and "RPTSTG" on page 61.
- For a list of the default settings for a specified country, see Table 28 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 C/C++ Programming Guide.

Examples

```c
/*Module/File Name: EDC3CTY */

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

int main(void) {
  _FEEDBACK fc;
  _INT4 function;
  _CHAR2 country;

  /* query the current country setting */
  function = 2; /* function 2 is query */
  CEE3CTY(&function,country,&fc);
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEE3CTY failed with message number %d\n", fc.tok_msgno);
    exit(2999);
  }

  /* if the current country is not Canada then set */
  /* it to Canada */
  if (memcmp(country,"CA",2) != 0) {
    memcpy(country,"CA",2);
    function = 1; /* function 1 is set */
    CEE3CTY(&function,country,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEE3CTY failed with message number %d\n", fc.tok_msgno);
      exit(2999);
    }
  }
}
```

Figure 15. C/C++ Example of CEE3CTY
CBL LIB,QUOTE
*Module/File Name: IGZT3CTY
**************************************************
** **
** Function: CEE3CTY - set default country **
** **
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3CTY.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 FUNCTN       PIC S9(9) BINARY.
  01 COUNTRY     PIC X(2).
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity   PIC S9(4) BINARY.
        04 Msg-No     PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl  PIC X.
      03 Facility-ID  PIC XXX.
    02 I-S-Info     PIC S9(9) BINARY.

PROCEDURE DIVISION.

Figure 16. COBOL Example of CEE3CTY (Part 1 of 2)
PARA-3CTYQRY.
************************************************
** Call CEE3CTY with the QUERY function,
** and display current country code.
************************************************
MOVE 2 TO FUNCTN.
CALL "CEE3CTY" USING FUNCTN, COUNTRY, FC.
IF CEE000 of FC THEN
   DISPLAY "THE CURRENT COUNTRY CODE IS "
   COUNTRY
ELSE
   DISPLAY "CEE3CTY(query) failed with msg "
   "Msg-No of FC UPON CONSOLE"
   STOP RUN
END-IF.

PARA-3CTYSET.
************************************************
** If the current country code is not the US,
** call CEE3CTY with the SET function to make
** it the US. Display result.
************************************************
IF ( COUNTRY IS NOT = "US" ) THEN
   MOVE 1 TO FUNCTN
   MOVE "US" TO COUNTRY
   CALL "CEE3CTY" USING FUNCTN,
         COUNTRY, FC
   IF CEE000 of FC THEN
      DISPLAY "THE NEW COUNTRY CODE IS ",
      COUNTRY
   ELSE
      DISPLAY "CEE3CTY(set) failed with msg "
      "Msg-No of FC UPON CONSOLE"
      STOP RUN
   END-IF
END-IF.

GOBACK.

Figure 16. COBOL Example of CEE3CTY (Part 2 of 2)
*PROCESS MACRO;
/* Module/File Name: IBM3CTY */
/*****************************************************************************/
/** **
/** Function: CEE3CTY - set current country **
/** **
/** In this example, a call is made to the query **
/** function of CEE3CTY to return the current **
/** default country setting. This is then **
/** printed out. If the current country code is **
/** not 'US', then it is set to 'US' and printed. **
/** **
/*****************************************************************************/
PLI3CTY: PROC OPTIONS(MAIN);

%INCLUDE CEEIBM4W;
%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 17. PL/I Example of CEE3CTY (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 CEE3DUMP 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 writeable.
- The directory found in environment variable TMPDIR (if the temporary directory is not /TMP).
- /TMP

(See the Fname suboption of CEE3DMP on page 132 for more information about the dump output filename.)

Dumps are written only in mixed-case U.S. English. 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:

Exception occurred during dump processing at nnnnnnnnn

nnnnnnnnnn 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.

If an abend occurs, or if any other condition occurs or is raised, the condition manager attempts to handle it. If the condition remains unhandled, and it is of
sufficient severity, the condition manager might, based on whether the
TERMTHDACT TRACE or DUMP suboption is specified, invoke dump services and
then terminate the program. You do not have to call 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.

The IBM-supplied default settings for the CEE3DMP options are:

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

**Syntax**

\[
\text{CEE3DMP}(-\text{title},-\text{options},-fc)
\]

*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 via 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)

Specifies the number of stack frames dumped from the call chain.

If STACKFRAME(ALL) is specified, all stack frames are dumped. No stack frame storage is dumped if STACKFRAME(0) is specified.

The particular information dumped for each stack frame depends on the VARIABLE, BLOCK, and STORAGE option declarations specified for CEE3DMP. The first stack frame dumped is the one associated with the routine that called CEE3DMP, followed by its caller, and proceeding backwards up the call chain.

PAGESize(n)

Specifies the number of lines on each page of the dump.

This value must be greater than 9. A value of 0 indicates that there should be no page breaks in the dump. The default setting is PAGESIZE(60).

FNAME(ddname)

Specifies the ddname of the file to which the dump report is written.

The ddname supplied in this option must be a valid ddname for the system on which the application is running. CEE3DMP does not check the ddname for validity, nor does CEE3DMP translate or modify the ddname. Supplying an invalid ddname can result in unpredictable behavior.

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

CONDition

Specifies that for each active condition on the call chain, the following information is dumped from the CIB:

- The address of the CIB.
- The message associated with the current condition token.
- The message associated with the original condition token, if different from the current one.
- The location of the error.
- The machine state at the time the condition manager was invoked, if an abend or hardware condition occurred.
• The abend code and reason code, if the condition occurred as a result of an abend.
• Language-specific error information.

The information supplied by Language Environment-conforming languages differs. PL/I supplies DATAFIELD, ONCHAR, ONCOUNT, ONFILE, ONKEY, and ONSOURCE built-in function (BIF) values, which are shown in the context of the condition raised.

This option does not apply to asynchronous signals.

**NOCONDition**

Does not dump condition information for active conditions on the call chain.

**ENTRY**

Includes in the dump a description of the routine that called CEE3DMP and the contents of the registers on entry to CEE3DMP.

**NOENTRY**

Does not include in the dump a description of the routine that called CEE3DMP and the contents of the registers on entry to CEE3DMP.

**GENOpts**

Includes in the dump a run-time options report like that generated by the use of the RUNOPTS 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 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>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**

• CICS consideration—Only ENCLAVE(CURRENT) and ENCLAVE(1) are supported on CICS.
• VM considerations—All values for the ENCLAVE option are supported on VM for nested enclaves created by the C system() function. Only ENCLAVE(CURRENT) and ENCLAVE(1) are supported on CMS for nested enclaves created by SVC LINK, CMSCALL or PL/I FETCH and CALL of a fetchable main.
• 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 hierarchic 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 71 for more information about the TERMTHDACT run-time option.
- For more information on generating dumps, see z/OS Language Environment Debugging Guide.
Examples

/*Module/File Name: EDC3DMP */

#include <stdio.h>
#include <leawi.h>
#include <stdlib.h>
#include <string.h>
#include <ceedcct.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)) {
        printf("CEE3DMP failed with msgno %d\n",
            fc.cek_msgno);
        exit(2999);
    }
}

Figure 18. C/C++ Example of CEE3DMP
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.

Figure 19. COBOL Example of CEE3DMP
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.

```plaintext
*PROCESS MACRO;
/* Module/File Name: IBM3DMP */
/*****************************************************************************/
/** Function: CEE3DMP - generate dump */
/** In this example, a call to CEE3DMP is made to */
/** request a dump of the run-time environment. */
/** Several options are specified, to customize */
/** the dump. */
/*****************************************************************************/
PLI3DMP: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL DMPTITL CHAR(80);
DCL OPTIONS CHARACTER (255);
DCL 01 FC, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

/* Specify a string to be printed at the top of */
/* each page of dump */
DMPTITL = 'This is the title for the dump report.';

/* Request that a traceback be provided, but */
/* no variables, stack frames, condition */
/* information, or registers be dumped */
OPTIONS = 'TRACE NOVAR SF(0) NOCOND NOENTRY';

/* Call CEE3DMP with options to customize dump */
CALL CEE3DMP (DMPTITL, OPTIONS, FC);
IF FBCHCK( FC, CEE000) THEN DO;
   PUT SKIP LIST( 'Successfully produced dump with'
      ' | | | | title "" | | DMPTITL | | "" ');
   PUT SKIP LIST( ' and options: ' | | OPTIONS );
   END;
ELSE DO;
   DISPLAY( 'CEE3DMP failed with msg '
      ' | | FC.MsgNo );
   STOP;
   END;
END PLI3DMP;
```

Figure 20. PL/I Example of CEE3DMP
CEE3GRC

Syntax

```c
CEE3GRC(--return_code--, fc--)
```

*return_code (output)*
The enclave return code.

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

The following 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 196 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

/*Module/File Name: EDC3GRC */

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

/*****************************************************/
/**
** Function: CEEHDLR - Register user condition handler
**
** : CEE3GRC - Get enclave return code
**
** 1. Register the user-written condition handler
**  CESETRC.
** 2. Call CERCDIV, which performs a divide-by-zero.
** 3. CESETRC is entered, sets the enclave return code to 999998 and resumes.
** 4. The main routine regains control and retrieves the enclave return code
**
*****************************************************/

void CERCDIV(int);

/*****************************************************/
Declaration of user-written condition handler
*****************************************************/

#ifndef __cplusplus
extern "C" {
#endif

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

#ifndef __cplusplus}
}
#endif

main()
{
  _INT4 idivisor = 0;
  _INT4 enclave_RC;
  _FEEDBACK feedback, new_feedback;
  _ENTRY pgmptr;
  _INT4 token;

  /*****************************************************/
  The condition handler CESETRC is registered
  *****************************************************/
  pgmptr.address = (_POINTER)&CESETRC;
  pgmptr.nesting = NULL;
  token = 97;
  CEEHDLR(&pgmptr, &token, &feedback);

Figure 21. C/C++ main() Routine that Calls CEEHDLR and CEE3GRC (Part 1 of 2)
Figure 21. C/C++ main() Routine that Calls CEEHDLR and CEE3GRC (Part 2 of 2)
/*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 22. C/C++ User-Written Condition Handler that Sets a User Enclave Return Code
/*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 23. C/C++ Subroutine that Generates the Divide-by-Zero Condition
CBL LIB,QUOTE,C,RENT,OPTIMIZE,NODYNAM
*Module/File Name: IGZT3GRC
**************************************************
** Module/File Name: IGZT3GRC
** CBL2GRC - Call the following Lang. Env. svcs: **
** : CEEHDLR - register user condition **
** handler **
** : CEE3GRC - get enclave return code **
**
** 1. Registers user condition handler CESETRC. **
** 2. Program then calls CERCDIV which performs **
**   a divide by zero operation. **
** 3. CESETRC gets control and set the enclave **
**   return code to 999998 and resumes. **
** 4. Regains control and retrieves the enclave **
**   return code. **
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3GRC.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 TOKEN PIC X(4).
  01 IDIVISOR PIC S9(9) BINARY VALUE ZERO.
  01 ENCLAVE-RC PIC S9(9) BINARY.
**
** Declares for condition handling **
**
  01 PGMPTR USAGE IS PROCEDURE-POINTER.
  01 FBCODE.
      02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.

Figure 24. COBOL Main Routine that Calls CEEHDLR and CEE3GRC (Part 1 of 2)
0001-BEGIN-PROCESSING.
*************************************************
** Register user condition handler CESETRC using  
**       CEEHDLR
*************************************************
   SET PGMPTR TO ENTRY "CESETRC".
   MOVE 97 TO TOKEN
   CALL "CEEHDLR" USING PGMPTR, TOKEN, FBCODE.
   IF NOT CEE000 of FBCODE THEN
      DISPLAY "CEEHDLR failed with msg "
      Msg-No of FBCODE UPON CONSOLE
      STOP RUN
   END-IF.
*************************************************
** Call CERCDIV to cause a divide by zero      
**    condition
*************************************************
   CALL "CERCDIV" USING IDIVISOR.
*************************************************
** Call CEE3GRC to get the enclave return code
*************************************************
   CALL "CEE3GRC" USING ENCLAVE-RC, FBCODE.
   IF NOT CEE000 of FBCODE THEN
      DISPLAY "CEEHDLR failed with msg "
      Msg-No of FBCODE UPON CONSOLE
      STOP RUN
   END-IF.
   IF (ENCLAVE-RC = 999998) THEN
      DISPLAY "Enclave return code "
      "set and retrieved."
   ELSE
      DISPLAY "*** Unexpected enclave return "
      "code of " ENCLAVE-RC " encountered"
   END-IF.
   GOBACK.
   End program CBL3GRC.

Figure 24. COBOL Main Routine that Calls CEEHDLR and CEE3GRC (Part 2 of 2)
DATA DIVISION.
  WORKING-STORAGE SECTION.
    01 ROUTINE PROCEDURE-POINTER.
    01 DENOMINATOR PIC S9(9) BINARY.
    01 NUMERATOR PIC S9(9) BINARY.
    01 RATIO PIC S9(9) BINARY.
    01 TOKEN PIC S9(9) BINARY VALUE 0.
    01 FC.
      02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
  REGISTER-HANDLER.
  *****************************************************
  ** Register handler
  *****************************************************
  SET ROUTINE TO ENTRY "CBL3SRC".
  CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
  IF NOT CEE000 OF FC THEN
    DISPLAY "CEEHDLR failed with msg "
    Msg-No OF FC UPON CONSOLE
    STOP RUN
  END-IF.

RAISE-CONDITION.
  *****************************************************
  ** Cause a zero-divide condition.
  *****************************************************
  MOVE 0 TO DENOMINATOR.
  MOVE 1 TO NUMERATOR.
  DIVIDE NUMERATOR BY DENOMINATOR
  GIVING RATIO.

Figure 25. COBOL Condition Handler that Sets a User Enclave Return Code and Resumes when a Divide-by-Zero Condition Occurs (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 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
        REDINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-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 25. COBOL Condition Handler that Sets a User Enclave Return Code and Resumes when a Divide-by-Zero Condition Occurs (Part 2 of 3)
PROCEDURE DIVISION USING CURRENT-CONDITION,
TOKEN, RESULT-CODE,
NEW-CONDITION.

HANDLE-CONDITION.

******************************************************************************
** Check for divide-by-zero condition (CEE349)                              
******************************************************************************

IF CEE349 of CURRENT-CONDITION THEN
  MOVE 761 TO ENCLAVE-RC
  CALL "CEE3SRC" USING ENCLAVE-RC, FEEDBACK
  IF NOT CEE000 of FEEDBACK THEN
    DISPLAY "CEE3SRC failed with msg "
    Msg-No of FEEDBACK UPON CONSOLE
  END-IF
END-IF.

SET PERCOLATE TO TRUE
GObACK.

END PROGRAM CBL3SRC.

Figure 25. COBOL Condition Handler that Sets a User Enclave Return Code and Resumes when a Divide-by-Zero Condition Occurs (Part 3 of 3)
CBL LIB,QUOTE,C,RENT,OPTIMIZE,NODYNAM
*Module/File Name: IGZTDIV
***********************************************************************
**Function : **
** A divide by zero is attempted. This **
** induces the invocation of user condition **
** handler CESETRC registered in program **
** CEGETRC. **
** : **
IDENTIFICATION DIVISION.
PROGRAM-ID. CERCDIV.

DATA DIVISION.
WORKING-Storage SECTION.
01 TO-DIVIDE    PIC S9(9) BINARY VALUE 1.
LINKAGE SECTION.
01 IDIVISOR     PIC S9(9) BINARY.

PROCEDURE DIVISION USING IDIVISOR.

**************************************************************************
** divide a constant by IDIVISOR. **
**************************************************************************
DIVIDE IDIVISOR INTO TO-DIVIDE.

GOBACK.
End program CERCDIV.

Figure 26. COBOL Subroutine that Generates a Divide-by-Zero
Figure 27. PL/I Example that Sets and Retrieves the User Enclave Return Code when a Divide-by-Zero is Generated (Part 1 of 2)
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--)

Figure 27. PL/I Example that Sets and Retrieves the User Enclave Return Code when a Divide-by-Zero is Generated (Part 2 of 2)
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

/*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 */
/* register condition handler */
token = 99;
routine.address = (_POINTER)&handler;
routine.nesting = NULL;
CEEHDLR(&routine,&token,&fc);
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
    exit (2999);
}

/* set up any condition sev 2 or higher */
c_1 = 3;
c_2 = 99;
cond_case = 1;
sev = 3;
control = 0;
memcpy(facid,"ZZZ",3);
isi = 0;

Figure 28. C/C++ Example of CEE3GRN (Part 1 of 2)
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 28. C/C++ Example of CEE3GRN (Part 2 of 2)
CBL LIB,QUOTE,NOOPT
*Module/File Name: IGZT3GRN
******************************************************************************
** **
** DRV3GRN - Drive sample program for CEE3GRN. **
** **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRV3GRN.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE            PROCEDURE-POINTER.
  01 DENOMINATOR        PIC S9(9) BINARY.
  01 NUMERATOR          PIC S9(9) BINARY.
  01 RATIO              PIC S9(9) BINARY.
  01 TOKEN              PIC S9(9) BINARY VALUE 0.
  01 FC.                COPY CEEIGZCT.
                         02 Condition-Token-Value.
                         COPY CEEIGZCT.
                         03 Case-1-Condition-ID.
                             04 Severity   PIC S9(4) BINARY.
                             04 Msg-No     PIC S9(4) BINARY.
                         03 Case-2-Condition-ID
                             REDEFINES Case-1-Condition-ID.
                             04 Class-Code PIC S9(4) BINARY.
                             04 Cause-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.

Figure 29. COBOL Example of CEE3GRN (Part 1 of 4)
REGISTER-HANDLER.
*************************************************
** Register handler
*************************************************
SET ROUTINE TO ENTRY "CBL3GRN".
CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEEHDLR failed with msg "
       Msg-No of FC UPON CONSOLE
   STOP RUN
END-IF.
RAISE-CONDITION.
*************************************************
** Cause a zero-divide condition.
*************************************************
MOVE 0 TO DENOMINATOR.
MOVE 1 TO NUMERATOR.
DIVIDE NUMERATOR BY DENOMINATOR
   GIVING RATIO.
UNREGISTER-HANDLER.
*************************************************
** UNregister handler
*************************************************
CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEEHDLU failed with msg "
       Msg-No of FC UPON CONSOLE
END-IF.
STOP RUN.
END PROGRAM DRV3GRN.

*************************************************
** **
** CBL3GRN - Call CEE3GRN to get the name of
** the routine that incurred
** the condition.
** **
*************************************************

Figure 29. COBOL Example of CEE3GRN (Part 2 of 4)
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3GRN.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 RNAME PIC X(80).
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
LINKAGE SECTION.
  01 TOKEN PIC S9(9) BINARY.
  01 RESULT PIC S9(9) BINARY.
  08 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 29. COBOL Example of CEE3GRN (Part 3 of 4)
PROCEDURE DIVISION USING CURCOND, TOKEN,
               RESULT, NEWCOND.

PARA-CBL3GRN.
   CALL "CEE3GRN" USING RNAME, FC.
   IF CEE000 of FC THEN
       DISPLAY "Name of routine which "
             "incurred the condition is: " RNAME
   ELSE
       DISPLAY "CEE3GRN failed with msg "
             Msg-No of FC UPON CONSOLE
       STOP RUN
   END-IF.

PARA-HANDLER.
***************************************************************
** In user handler - resume execution
***************************************************************
   SET RESUME TO TRUE.
   GOBACK.

END PROGRAM CBL3GRN.

Figure 29. COBOL Example of CEE3GRN (Part 4 of 4)
CEE3GRO—Get offset of condition

The CEE3GRO service returns the offset within a failing routine of the most recent condition. If there are nested conditions, the most recently signaled condition is returned.

```
*PROCESS OPT(0), MACRO;
/* Module/File Name: IBM3GRN */
*******************************************************************************/
/**
/** Function: CEE3GRN - example of CEE3GRN **/
/** invoked from PL/I ON-unit **/
*******************************************************************************/

IBM3GRN: PROCEDURE OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DECLARE
  RNAME CHAR(80),  /* Feedback token */
    FC,
    /* Feedback token */
    MsgSev REAL FIXED BINARY(15,0),
    MsgNo  REAL FIXED BINARY(15,0),
    Flags,  /* Feedback token */
    Case   BIT(2),
    Severity BIT(3),
    Control BIT(3),
    FacID CHAR(3),  /* Facility ID */
    ISI    /* Instance-Specific Information */
      REAL FIXED BINARY(31,0),
  divisor FIXED BINARY(31) INITIAL(0);

ON ZERODIVIDE BEGIN;
  /* Call CEE3GRN to get the name of the routine */
  /* that incurred the most recently signalled */
  /* condition */

    CALL CEE3GRN ( RNAME, FC );
    IF FBCHCK( FC, CEE000 ) THEN DO;
      PUT SKIP LIST( 'The most recently signalled' );
      'condition was incurred by '|| RNAME );
      END;
    ELSE DO;
      DISPLAY( 'CEE3GRN failed with msg' );
      FC.MsgNo );
      END;

    END /* ON ZeroDivide */;

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

END IBM3GRN;
```

Figure 30. PL/I Example of CEE3GRN

CEE3GRO—Get offset of condition

The CEE3GRO service returns the offset within a failing routine of the most recent condition. If there are nested conditions, the most recently signaled condition is returned.
Syntax

```c
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:

<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

CBL LIB,QUOTE,NOOPT
*Module/File Name: IGZT3GRO
***********************************************************************
** **
** DRV3GRO - Register a condition handler **
** that calls CEE3GRO to determine **
** the offset in the program that **
** incurred the condition. **
** **
***********************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRV3GRO.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 ROUTINE PROCEDURE-POINTER.
01 DENOMINATOR PIC S9(9) BINARY.
01 NUMERATOR PIC S9(9) BINARY.
01 RATIO PIC S9(9) BINARY.
01 TOKEN PIC S9(9) BINARY VALUE 0.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.

Figure 31. COBOL Example of CEE3GRO (Part 1 of 4)
REGISTERS-HANDLER.
******************************************************************************
** Register handler ******
******************************************************************************
SET ROUTINE TO ENTRY "CBL3GRO".
CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
  DISPLAY "CEEHDLR failed with msg "
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.
RAISE-CONDITION.
******************************************************************************
** Cause a zero-divide condition ******
******************************************************************************
MOVE 0 TO DENOMINATOR.
MOVE 1 TO NUMERATOR.
DIVIDE NUMERATOR BY DENOMINATOR
  GIVING RATIO.
UNREGISTERS-HANDLER.
******************************************************************************
** Unregister handler ******
******************************************************************************
CALL "CEEHDLU" USING ROUTINE, FC.
IF NOT CEE000 of FC THEN
  DISPLAY "CEEHDLU failed with msg "
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.
STOP RUN.
END PROGRAM DRV3GRO.
******************************************************************************
** CBL3GRO - Call CEE3GRO to get the offset ******
** in the routine that incurred the condition. ******
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3GRO.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 ROFFSET PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
         03 Case-1-Condition-ID.
            04 Severity PIC S9(4) BINARY.
            04 Msg-No PIC S9(4) BINARY.
         03 Case-2-Condition-ID
            REDEFINES Case-1-Condition-ID.
            04 Class-Code PIC S9(4) BINARY.
            04 Cause-Code PIC S9(4) BINARY.
         03 Case-Sev-Ctl PIC X.
         03 Facility-ID PIC XXX.
         02 I-S-Info PIC S9(9) BINARY.

LINKAGE SECTION.
01 TOKEN PIC S9(9) BINARY.
01 RESULT PIC S9(9) BINARY.
08 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 31. COBOL Example of CEE3GRO (Part 3 of 4)
PROCEDURE DIVISION USING CURCOND, TOKEN, RESULT, NEWCOND.

PARA-CBL3GRO.
   CALL "CEE3GRO" USING ROFFSET, FC.
   IF CEE000 of FC THEN
      DISPLAY "Offset in routine which "
      "incurred the condition is: "
      ROFFSET
   ELSE
      DISPLAY "CEE3GRO failed with msg "
      Msg-No of FC UPON CONSOLE
   END-IF.

PARA-HANDLER.
********************************************************************
** In user handler - resume execution
********************************************************************
   SET RESUME TO TRUE.
   GOBACK.
END PROGRAM CBL3GRO.

Figure 31. COBOL Example of CEE3GRO (Part 4 of 4)

*Process lc(101),opt(0),s,map,list,stmt,a(f),ag;
*Process macro;
DRV3GRO: Proc Options(Main);

/Module/File Name: IBM3GRO /*
/**************************************************************************
** ** DRV3GRO - Register a condition handler that **
** ** calls CEE3GRO to determine **
** ** the offset in the routine that **
** ** incurred the condition. **
** ***************************************************************************/

%include CEEIBMCT;
%include CEEIBMAW;
declare 01 FBCODE feedback;
declare DENOMINATOR real fixed binary(31,0);
declare NUMERATOR real fixed binary(31,0);
declare RATIO real fixed binary(31,0);
declare PLI3GRO external entry;
declare U_PTR pointer;
declare 01 U_DATA,
   03 U_CNTL fixed binary(31,0),
   03 U_TOK pointer;

U_PTR = addr(U_DATA);
U_CNTL = 0;

/* Set Resume Point */

Figure 32. PL/I Example of CEE3GRO (Part 1 of 3)
Display('Setting resume point via CEE3SRP');
Call CEE3SRP(U_TOK,FBCODE);
Display('After call to CEE3SRP ... Resume point');
If U_CNTL = 0
  Do;
    Display('First time through...');
    /* Register User Handler */
    Display('Registering user handler');
    Call CEEHDLR(PLI3GRO, U_PTR, FBCODE);
    If FBCHECK(FBCODE, CEE000) then
      Do;
        /* Cause a zero-divide condition */
        DENOMINATOR = 0;
        NUMERATOR = 1;
        RATIO = NUMERATOR/DENOMINATOR;
      End;
    Else
      Do;
        Display('CEEHDLR failed with msg');
        Display(MsgNo);
      End;
    End;
Else
  Do;
    Display('2nd time...User can do whatever');
    /* Unregister handler */
    Call CEEHDLU(PLI3GRO, FBCODE);
    If FBCHECK (FBCODE, CEE000) Then
      Display('Main: unregistered PLI3GRO');
    Else
      Do;
        Display('CEEHDLU failed with msg');
        Display(MsgNo);
      End;
    End
End DRV3GRO;

Figure 32. PL/I Example of CEE3GRO (Part 2 of 3)
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.

Figure 32. PL/I Example of CEE3GRO (Part 3 of 3)
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 function is specified as:</th>
<th>Then desired_language:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 or 3</td>
<td>Contains the desired national language identification. In this case, it is an input parameter. Table 23 on page 171 contains a list of national language identifiers. <strong>Note:</strong> Language Environment supports only these national languages:</td>
</tr>
<tr>
<td></td>
<td>ENU  MIXED-Case U.S. English</td>
</tr>
<tr>
<td></td>
<td>UEN  UPPERCASE U.S. English</td>
</tr>
<tr>
<td></td>
<td>JPN  Japanese.</td>
</tr>
<tr>
<td>2</td>
<td>Returns the current language on top of the stack. In this case, it is an output parameter.</td>
</tr>
<tr>
<td>4</td>
<td>Returns the discarded national language. In this case, it is an output parameter.</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>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 in the</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**

- **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 CEE3LNG callable service. CEE3LNG 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 CEE3LNG and either CEESETL or setlocale().

- 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.

For more information

- See “NATLANG” on page 51 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 443.
- For more information on setlocale(), see z/OS C/C++ Programming Guide.
Examples

/*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 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 33. C/C++ Example of CEE3LNG
**Module/File Name: IGZT3LNG**

**************************************************
** CBL3LNG - Set national language **
** In this example, CEE3LNG is called to query **
** the current national language setting. If **
** the setting is not mixed-case U.S. English, **
** CEE3LNG is called to change the setting. **
**
**************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3LNG.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 FUNCTN PIC S9(9) BINARY.
  01 LANG PIC X(3).
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.

---

Figure 34. COBOL Example of CEE3LNG (Part 1 of 2)
PARA-3LNGQRY.
===============================================================
** Specify 2 for QUERY function.
** Call CEE3LNG to query the current
** national language setting
===============================================================

MOVE 2 TO FUNCTN.
CALL "CEE3LNG" USING FUNCTN, LANG, FC.
  IF CEE000 of FC THEN
    DISPLAY "Current National Language is: "
      LANG
  ELSE
    DISPLAY "CEE3LNG(query) failed with msg "
      Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.

PARA-3LNGSET.
===============================================================
** If the current national language is not
** mixed-case U.S. English, then call
** CEE3LNG with the SET function (1) to
** change the national language to mixed-case
** U.S. English
===============================================================

IF ( LANG IS NOT = "ENU" ) THEN
  MOVE 1 TO FUNCTN
  CALL "CEE3LNG" USING FUNCTN, LANG, FC
  IF NOT CEE000 of FC THEN
    DISPLAY "CEE3LNG(set) failed with msg "
      Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF
  DISPLAY "The national language has ",
    "been changed to mixed-case "
    "U.S. English (ENU)."
END-IF.

GOBACK.

---

Figure 34. COBOL Example of CEE3LNG (Part 2 of 2)
PROCESS MACRO;
/* Module/File Name: IBM3LNG */
/******************************/
/** */
/** Function: CEE3LNG - set national language */
/** */
/** In this example, CEE3LNG is called to query the */
/** current national language setting. If the */
/** setting is not mixed case American English, */
/** CEE3LNMG is called to change the setting to that */
/** */
/********************************/
PLI3LNG: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL FUNCTN REAL FIXED BINARY(31,0);
DCL LANG CHARACTER (3);
DCL 01 FC,         /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo  REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID  CHAR(3),     /* Facility ID */
   03 ISI    /* Instance-Specific Information */
         REAL FIXED BINARY(31,0);

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

/* Call CEE3LNG with function code 2 to query */
/* national language */

CALL CEE3LNG (FUNCTN, LANG, FC);
IF FBCHECK(FC, CEE000) THEN DO;
   PUT SKIP LIST('The current national language ' || 'is ' || LANG);
END;
ELSE DO;
   DISPLAY('CEE3LNG failed with msg ' || FC.MsgNo);
   STOP;
END;

Figure 35. PL/I Example of CEE3LNG (Part 1 of 2)
/* If the current language is not mixed-case */
/* American English, set it to mixed-case */
/* American English */

IF LANG = 'ENU' THEN DO;
   FUNCTN = 1;
   CALL CEE3LNG ( FUNCTN, 'ENU', FC);
   IF ~ FBCHECK( FC, CEE000) THEN DO;
      DISPLAY( 'CEE3LNG failed with msg'
               || FC.MsgNo );
      STOP;
   END;
   CALL CEE3LNG ( 2, LANG, FC);
   IF FBCHECK( FC, CEE000) THEN DO;
      PUT SKIP LIST('The national language is now'
                     || LANG);
   END;
   ELSE DO;
      DISPLAY( 'CEE3LNG failed with msg'
               || FC.MsgNo );
      STOP;
   END;
END /* Language is not ENU */;

END PLI3LNG;

Figure 35. PL/I Example of CEE3LNG (Part 2 of 2)

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>
</tbody>
</table>
Table 23. National Language Codes (continued)

<table>
<thead>
<tr>
<th>ID</th>
<th>National Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLD</td>
<td>Dutch</td>
</tr>
<tr>
<td>NLB</td>
<td>Belgian Dutch</td>
</tr>
<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 28 on page 515.

**Syntax**

```plaintext
CEE3MCS(country_code, currency_symbol, fc)
```

`country_code (input)`

A 2-character fixed-length string representing one of the country codes found in Table 28 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 an invalid country_code, 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.

For more information

- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 120 for an explanation of the CEE3CTY callable service.

Examples

```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,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:" u %.2s\n",currency);
}
```

Figure 36. C/C++ Example of CEE3MCS
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(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
      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 37. COBOL Example of CEE3MCS
CEE3MDS—Get default decimal separator

CEE3MDS returns the default decimal separator for the country specified by `country_code`. For a list of the default settings for a specified country, see Table 28 on page 515.

**Syntax**

```
$CEEMDS(country_code,decimal_separator,fc)
```
**country_code (input)**

A 2-character fixed-length string representing one of the country codes found in Table 28 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.

**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 <code>'decimal_separator'</code> was truncated and was not defined in CEE3MDS.</td>
</tr>
<tr>
<td>CEE3C5</td>
<td>2</td>
<td>3461</td>
<td>The country code <code>country-code</code> was invalid for CEE3MDS. The default decimal separator <code>'decimal_separator'</code> was returned.</td>
</tr>
</tbody>
</table>

**Usage notes**

- If you specify an invalid `country_code`, 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 120](#) for an explanation of the CEE3CTY callable service.
```c
#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);
```
CBL LIB,QUOTE
*Module/File Name: IGZT3MDS
*****************************************************************************
**
** CBL3MDS - Call CEE3MDS to get the **
** default decimal separator **
**
*****************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3MDS.

DATA DIVISION.
WORKING-STORAGE SECTION.
 01 COUNTRY PIC X(2).
 01 DECSEP PIC X(2).
 01 FC.
   02 Condition-Token-Value.
   COPY CEEIGZCT.
     03 Case-1-Condition-ID.
       04 Severity PIC S9(4) BINARY.
       04 Msg-No PIC S9(4) BINARY.
     03 Case-2-Condition-ID
       REDEFINES Case-1-Condition-ID.
       04 Class-Code PIC S9(4) BINARY.
       04 Cause-Code PIC S9(4) BINARY.
     03 Case-Sev-Ctl PIC X.
     03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
*****************************************************************************
** Specify the country code for the US in the **
** call to CEE3MDS. **
** If call was successful, print result. **
*****************************************************************************
PARA-CBL3MDS.
  MOVE "US" TO COUNTRY.
  CALL "CEE3MDS" USING COUNTRY, DECSEP, FC.
  IF CEE000 of FC THEN
    DISPLAY "The default Decimal Separator "
    "for " COUNTRY " is " DECSEP ""
  ELSE
    DISPLAY "CEE3MDS failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
  GOBACK.

Figure 40. COBOL Example of CEE3MDS
CEE3MTS—Get Default thousands separator

CEE3MTS returns the default thousands separator for the specified country with `country_code`.

**Syntax**

```plaintext
>>>CEE3MTS—(—country_code—,—thousands_separator—,—fc—)
```
country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 28 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.

thousands_separator (output)
A 2-character fixed-length string representing the default thousands separator for the country specified. The thousands separator is left-justified and padded on the right with a blank.

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

The following 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 an invalid country_code, 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 28 on page 515.
- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 120 for an explanation of the CEE3CTY callable service.
Examples

/*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 , CEE800 ) != 0 ) {
    printf("CEE3MTS failed with message number %d\n", fc.tok_msgno);
    exit(2999);
  }
  /* print out the default thousands separator */
  printf("The default thousands separator for Canada");
  printf(" is: %.2s\n",thousand);
}

Figure 42. C/C++ Example of CEE3MTS
CBL LIB,QUOTE
*Module/File Name: IGZT3MTS
******************************************************************************
** **
** CBL3MTS - Call CEE3MTS to obtain the **
** default thousands separator **
********************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3MTS.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 COUNTRY PIC X(2).
  01 THOUSEP PIC X(2).
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-Value.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID.
          REDEFINES Case-1-Condition-Value.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-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 CEE0000 of FC THEN
    DISPLAY "The default Thousands Separator"
    " for " COUNTRY " is " THOUSEP ""**
  ELSE
    DISPLAY "CEE3MDS failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
  GOBACK.

Figure 43. COBOL 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

```plaintext
>>CEE3PRM—(char_parm_string,—fc—)
```
**CEE3PRM**

*char_parm_string (output)*

An 80-byte fixed-length string passed by CEE3PRM.

On return from this service, the *char_parm_string* contains the parameter string specified at invocation of the program. If this parameter string is longer than 80 characters, it is truncated. If the parameter string is shorter than 80 characters, the returned string is padded with blanks. If the program argument passed to the service is absent, or is not a character string, *char_parm_string* is blank.

*fc (output)*

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

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.

**Examples**

```c
/*Module/File Name: EDC3PRM */

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

int main() {

_CHAR80 parm;
_FEEDBACK fc;

CEE3PRM(parm,&fc);
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {  
    printf("CEE3PRM failed with message number %d\n", fc.tok_msgno);
    exit(2999);
}  
printf("%.80s\n",parm);
}
```

*Figure 45. C/C++ Example of CEE3PRM*
CBL LIB,QUOTE
*Module/File Name: IGZT3PRM
**************************************************
** **
** CBL3PRM - Call CEE3PRM to query the **
** parameter string **
** **
** In this example, a call is made to **
** CEE3PRM to return the parameter string **
** that was specified at invocation of the **
** program. The string is then displayed. **
** **
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3PRM.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 PARMSTR PIC X(80).
01 FC.
   02 Condition-Token-Value.
   COPY CEEIGZCT.
   03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
   03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
   03 Case-Sev-Ctl PIC X.
   03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
PARA-CBL3PRM.
   CALL "CEE3PRM" USING PARMSTR, FC.
   IF CEE000 THEN
      DISPLAY "Program arguments specified:"
      "*** PARMSTR ***"
   ELSE
      DISPLAY "CEE3PRM failed with msg "
      Msg-No of FC UPON CONSOLE
      STOP RUN
   END-IF.
   GOBACK.

Figure 46. COBOL Example of CEE3PRM
CEE3RPH

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

```plaintext
>>>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>Feedback Code (fc)</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 61 for more information about the RPTSTG run-time option.
- See "RPTOPTS" on page 60 for more information about the RPTOPTS run-time option.
- See PL/I for MVS & VM Programming Guide for further information on PLIXHD.

Examples

```c
/*Module/File Name: EDC3RPH */
#pragma runopts(RPTOPTS(ON))
#include <string.h>
#include <leawi.h>
#include <ceeedcct.h>

int main(void) {
    _CHAR80 heading;
    /* initialize heading to blanks and then set the */
    /* heading */
    memset(heading,' ',80);
    memcpy(heading,"User Defined Report Heading",27);
    /* set the report heading...do not need to check */
    /* feedback token because all return codes are */
    /* successful */
    CEE3RPH(heading,NULL);
    /* .
     . */
}
```

Figure 48. C/C++ Example of CEE3RPH
CBL LIB,QUOTE
*Module/File Name: IGZT3RPH
*********************************************************
**  **
** CBL3RPH - Call CEE3RPH to set report heading**
**  **
** In this example, a call is made to CEE3RPH **
** to set the report heading that appears at **
** the top of each page of the options report **
** (generated by RPTOPTS) and storage report **
** (generated by RPTSTG). **
**  **
*********************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3RPH.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 RPTHEAD PIC X(80).
01 FC.
   02 Condition-Token-Value.
     COPY CEEIGZCT.
     03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
     03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
     03 Case-Sev-Ctl PIC X.
     03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.

*********************************************************
** Specify user-defined report heading via CEE3RPH**
*********************************************************
PARA-CBL3RPH.
   MOVE "My options and storage reports heading" TO RPTHEAD.
   CALL "CEE3RPH" USING RPTHEAD, FC.
   IF NOT CEE000 OF FC THEN
      DISPLAY "CEE3RPH failed with msg "
      Msg-No OF FC UPON CONSOLE
      STOP RUN
   END-IF.
   GOBACK.

Figure 49. COBOL Example of CEE3RPH
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.

Figure 50. PL/I Example of CEE3RPH
Pop the pushed settings of the Language Environment conditions from the condition stack using an action value of 4 (POP). This reinstates the previous condition settings as the current condition settings.

Push the current settings of the Language Environment conditions on to the Language Environment-managed condition stack and alter the settings of the Language Environment conditions to those supplied by the caller.

The enabled or disabled Language Environment conditions are:

- **fixed-overflow**: When enabled, raises the fixed-overflow condition when an overflow occurs during signed binary arithmetic or signed left-shift operations.

- **decimal-overflow**: When enabled, raises the decimal-overflow condition when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the results.

- **underflow**: When enabled, raises the underflow condition when the result characteristic of a floating-point operation is less than zero and the result fraction is not zero. For an extended-format floating-point result, the condition is raised only when the high-order characteristic underflows.

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

When you use the CEE3SPM callable service, maintenance of the condition stack is required. For example, one user-written condition handler can disable a hardware condition while another enables it. Therefore, do not assume that the program mask is at a given setting. The program mask is set differently based on different HLL requirements. You can find out what the current setting is by using the QUERY function of CEE3SPM.

Language Environment initialization sets conditions based on the languages in the initial load module. Each language present adds to the conditions that are enabled.

Some S/370 hardware interrupt codes and their matching Language Environment feedback codes appear in Table 24 on page 192.

---

### Syntax

```plaintext
CEE3SPM(action, cond_string, fc)
```

**action (input)**
The action to be performed. action is specified as a fullword binary signed integer corresponding to one of the numbers in the following list:

1—SET

Alters the settings of the Language Environment conditions to those specified in the cond_string parameter.

2—QUERY

Queries the current settings of the Language
Environment conditions and return the settings in the cond_string parameter.

3—PUSH

Pushes the current settings of the Language Environment conditions on to the Language Environment-managed condition stack.

4—POP

Pops the pushed settings of the Language Environment conditions from the condition stack, discarding the current settings, and reinstating the previous condition settings as the current condition settings.

5—PUSH, SET

Pushes the current settings of the Language Environment conditions on to the Language Environment-managed condition stack and alters the settings of the Language Environment conditions to those supplied by the caller in the cond_string parameter.

cond_string (input/output)

A fixed-length string of 80 bytes containing a sequence of identifiers representing the requested settings for the Language Environment conditions that can be enabled and disabled.

A list of conditions enabled and disabled and their associated identifiers is given below:

<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>
</tbody>
</table>
## CEE3SPM

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<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

/*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 \
", fd.tok_msgno);
        exit(2999);
    }
}

Figure 51. C/C++ Example of CEE3SPM
CBL LIB,QUOTE
*Module/File Name: IGZT3SPM
***********************************************************************
** **
** CBL3SPM - Call CEE3SPM to query and modify **
** Lang. Environ. hardware condition **
** enablement **
** In this example, a call is made to CEE3SPM **
** to check the setting of the program mask. **
** See the parameter list of CEE3SPM to **
** interpret what is returned as CONDSTR in **
** this example. **
**
***********************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBL3SPM.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ACTION PIC S9(9) BINARY.
  01 CONDSTR PIC X(80).
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.

******************************************************************************
** Specify 2 for the QUERY function.
** Pass ACTION in the call to CEE3SPM to return
** the condition string DISPLAY results.
******************************************************************************
PARA-CBL3SPM.
  MOVE 2 TO ACTION.
  CALL "CEE3SPM" USING ACTION, CONDSTR, FC.
  IF CEE000 of FC THEN
    DISPLAY "The current setting of the ",
    "program mask is: " CONDSTR
  ELSE
    DISPLAY "CEE3SPM failed with msg "
    Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
  GOBACK.

Figure 52. COBOL Example of CEE3SPM
*PROCESS MACRO;
/* Module/File Name: IBM3SPM */
/***************************************************************************/
/** Function: CEE3SPM - Query and Modify LE/370 */
/** Hardware Condition Enablement */
/** */
/** This example calls CEE3SPM to query the current */
/** setting of the program mask. See the parameter */
/** list of CEE3SPM to interpret what is returned */
/** as CONDSTR in this example. */
/** */
/***************************************************************************/
PLI3SPM: PROC OPTIONS(MAIN);

INCLUDE CEEIBMAW;
INCLUDE CEEIBMCT;

DCL ACTION REAL FIXED BINARY(31,0);
DCL CONDSTR CHAR(80);
DCL 01 FC, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);

/* Call CEE3SPM to query the current setting of */
/* the program mask */

ACTION = 2; /* Specify action code 2 to query */
/* the program mask */
CALL CEE3SPM (ACTION, CONDSTR, FC);
IF FBCHECK( FC, CEE000) THEN DO;
  PUT SKIP LIST('The initial setting of the program mask is:
    || CONDSTR );
END;
ELSE DO;
  DISPLAY('CEE3SPM failed with msg ' || FC.MsgNo);
  STOP;
END;

Figure 53. 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**

```
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.

For more information
- See [z/OS Language Environment Programming Guide](../doc/zoes) for more information about the CEE3SRC callable service.

Examples
CEE3SRC is used with CEE3GRC—see the examples for CEE3GRC, "Examples" on page 137.

**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
```assembly
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:
  ```assembly
  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 consideration—A Service Label compiler directive must be specified immediately after the call to CEE3SRP.

Examples
For examples of how to use CEE3SRP in combination with other CEEMRCE and CEEHDLR, see “CEEMRCE—Move resume cursor explicit” on page 375.

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

```cpp
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

/ *Module/ File Name: EDC3USR  * /

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

typedef struct {
    int value1, value2, value3;
    char slot1_80;
} info_struct;

int main (void) {
    _INT4 function_code, field_number, field_value;
    _FEEDBACK fc;
    info_struct *info;

    info = (info_struct *)malloc(sizeof(info_struct));
    /*
     .
     . */
    /* Set User field 1 to point to info_struct */
    function_code = 1;
    field_number = 1;
    field_value = (int)info;

    CEE3USR(&function_code, &field_number, &field_value, &fc);
    if (_FBCHECK (fc, CEE000) != 0) {
        printf("CEE3USR failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }
    /*
     .
     . */
    /* get the value of field 2 */
    function_code = 2;
    field_number = 1;

    CEE3USR(&function_code, &field_number, &field_value, &fc);
    if (_FBCHECK (fc, CEE000) != 0) {
        printf("CEE3USR failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }
    /*
     .
     . */
}

Figure 54. C/C++ Example of CEE3USR
**Module/ File Name: IGZT3USR**

**************************************************

** CBL3USR - Call CEE3USR to set or query user **

** area fields **

** In this example, CEE3USR is called twice: **

** once to set the value of a user area, and **

** once to query it. **

**************************************************

**IDENTIFICATION DIVISION.**

**PROGRAM-ID. CBL3USR.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**01 FUNCODE PIC S9(9) BINARY.**

**01 FIELDNO PIC S9(9) BINARY.**

**01 INVALUE PIC S9(9) BINARY.**

**01 FC.**

**02 Condition-Token-Value.**

**COPY CEEIGZCT.**

**03 Case-1-Condition-ID.**

**04 Severity PIC S9(4) BINARY.**

**04 Msg-No PIC S9(4) BINARY.**

**03 Case-2-Condition-ID**

**REDEFINES Case-1-Condition-ID.**

**04 Class-Code PIC S9(4) BINARY.**

**04 Cause-Code PIC S9(4) BINARY.**

**03 Case-Sev-Ctl PIC X.**

**03 Facility-ID PIC XXX.**

**02 I-S-Info PIC S9(9) BINARY.**

**PROCEDURE DIVISION.**

**************************************************

** Specify 1 for SET function.**

** Specify field number 1 to set the value field**

** number 1.**

** Specify 23 to make the value of field number 1**

** equal to 23.**

**************************************************

**PARA-3USRSET.**

**MOVE 1 TO FUNCODE.**

**MOVE 1 TO FIELDNO.**

**MOVE 23 TO INVALUE.**

**CALL "CEE3USR" USING FUNCODE, FIELDNO,**

**INVALUE, FC.**

**IF NOT CEE000 of FC THEN**

**DISPLAY "CEE3USR failed with msg "**

**Msg-No of FC UPON CONSOLE**

**STOP RUN**

**END-IF.**

---

*Figure 55. COBOL Example of CEE3USR (Part 1 of 2)*
** Specify 2 for QUERY function.
** Specify field number 1 to query the value
** of field number 1.
******************************************************************************
PARA-3USRQRY.
    MOVE 2 TO FUNCODE.
    MOVE 1 TO FIELDNO.
    CALL "CEE3USR" USING FUNCODE, FIELDNO, INVALUE, FC.
    IF CEE000 of FC THEN
        DISPLAY "User Area field " FIELDNO " is: " INVALUE
    ELSE
        DISPLAY "CEE3USR failed with msg "
        Msg-No of FC UPON CONSOLE
        STOP RUN
    END-IF.
    GOBACK.
Figure 55. COBOL Example of CEE3USR (Part 2 of 2)

*PROCESS MACRO;
/* Module/File Name: IBM3USR */
/***************************************************************************/
/** Function: CEE3USR - set/query user area fields **/  
/** In this example, CEE3USR is called twice: once **/  
/** to set the value of a user area, and once to **/  
/** query it. **/  
******************************************************************************/
PLI3USR: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMMAW;
%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,  
    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 */

Figure 56. PL/I Example of CEE3USR (Part 1 of 2)
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 CEESCE0N 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 29 on page 519 for a list of valid picture character terms that can be specified in input_char_date.

picture_string (input)

A halfword length-prefixed character string (VSTRING), indicating the format of the date specified in input_char_date.

Each character in the picture_string corresponds to a character in input_char_date. For example, if you specify MMDDYY as the picture_string, CEECBLDY reads an input_char_date of 060288 as 02 June 1988.

If delimiters such as the slash (/) appear in the picture string, leading zeros can be omitted. For example, the following calls to CEECBLDY:

```
MOVE '6/2/88' TO DATEVAL.
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);
```

would each assign the same value, 148155 (02 June 1988), to COBINTDATE.

Whenever characters such as colons or slashes are included in the picture_string (such as HH:MM:SS YY/MM/DD), they count as placeholders but are otherwise ignored.

See Table 29 on page 519 for a list of valid picture character terms and Table 30 on page 520 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 30 on page 520 for an additional example. See also Table 31 on page 520 for a list of Japanese Eras supported by CEEDATE.

If `picture_string` includes era symbol `<CCCC>` or `<CCCCCCC>`, the `YY` position in `input_char_date` is replaced by the year number within the era. See Table 30 on page 520 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 <code>&lt;JJJJ&gt;</code> or <code>&lt;CCCC&gt;</code> 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 Programming Guide](https://www.ibm.com/support/docview.wss?uid=swg21201743) or [COBOL for OS/390 & VM Programming Guide](https://www.ibm.com/support/docview.wss?uid=swg21201740) 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 421 for more information about the CEESCEN callable service.

- See Table 29 on page 519 for a list of valid picture character terms that can be specified in `input_char_date`. 
Examples

CELB LIB,QUOTE
*Module/ File Name: IGZTBLD
******************************************************************************
** 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 CHRDTE.
   02 Vstring-length PIC S9(4) BINARY.
   02 Vstring-text PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of CHRDTE.
01 PICSTR.
   02 Vstring-length PIC S9(4) BINARY.
   02 Vstring-text 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 CEEIIGZCT.
   03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
   03 Case-2-Condition-ID REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-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.

Figure 57. 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. CEECMII 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 CEECMII to copy any insert information into the ISI associated with the condition.
Syntax

```plaintext
CALL 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 or insert 2). It corresponds to an insert number specified with an `ins` tag in the message source file created by the CEEBLDTX EXEC.

`insert_data` *(input)*

A halfword-prefixed length string that represents the insert data. The entire length described in the halfword prefix is used without truncation. DBCS strings must be enclosed within shift-out (`X'0E'`) and shift-in (`X'0F'`) characters.

The maximum size for an individual insert data item is 254 bytes.

`fc` *(output)*

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

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 <code>message-number</code> and facility ID <code>facility-id</code> 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 <code>I_S_info</code> 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 <code>message-number</code> and facility ID <code>facility-id</code> 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 50 for more information about the MSGQ run-time option.
Examples

/*Module/File Name: EDCCMI */

/**************************************************
**
** FUNCTION: CEENCOD - set up a condition token  
** : CEECMI - store and load message  
**      insert data  
** : CEEMSG - retrieve, format, and  
**      dispatch a message to  
**      message file  
**
** This example illustrates the invocation of  
** the Lang. Environ. message services to  
** store and load message insert data.  
** The resulting message and insert is written  
** to the Lang. Environ. MSGFILE ddname.  
**
**************************************************/

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

void main ()
{
    _INT2 c_1,c_2,cond_case,sev,control;
    _CHAR3 facid;
    _INT4 isi;
    _VSTRING insert;
    _FEEDBACK ctok;
    _FEEDBACK fbcode;
    _INT4 MSGFILE;
    _INT4 insert_no;

    insert.length = 18;
    memcpy(insert.string ,"CEPGCMI's insert",insert.length);
    /*give ctok value of hex 0000000A40E7D40700000000 */
    /*sev = 0 msgno = 10 facid = XMP */
    c_1 = 0;
    c_2 = 10;
    cond_case = 1;
    sev = 0;
    control = 0;
    memcpy(facid,"XMP",3);
    isi = 0;
}

Figure 58. C/C++ Example of CEECMI (Part 1 of 2)
Call CEENCOD to set-up a condition token

CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
facid,&Isi,&ctok,&fbcode);
if (_FBCHECK ( fbcode , CEE000 ) != 0 )
  printf("CEENCOD failed with message number %d\n",
    fbcode.tok_msgno);

Call CEECMI to create a message insert

CEECMI(&ctok, &insert_no, &insert, &fbcode);
if ( _FBCHECK ( fbcode , CEE000 ) != 0 )
  printf("CEECMI failed with message number %d\n",
    fbcode.tok_msgno);

Call CEEMSG to issue the message

CEEMSG(&ctok, &MSGFILE, &fbcode);
if ( _FBCHECK ( fbcode , CEE000 ) != 0 )
  printf("CEEMSG failed with message number %d\n",
    fbcode.tok_msgno);

Figure 58. C/C++ Example of CEECMI (Part 2 of 2)
**Module/File Name: IGZTCMI**

**********************************************
** Function: CEECMI - Store and load message *
**     insert data *
** : CEENCOD - Construct a condition *
**     token *
** : CEEMSG - Dispatch a Message. *
**
** This example illustrates the invocation *
** of the Lang. Environ. message services to *
** store and load message insert data. *
** CEENCOD is called to construct a token *
** for a user defined message (message 10) *
** in a user message file. *
** CEECMI is called to insert text into *
** message 10. The resulting message and *
** insert is written to the MSGFILE. *
**
**********************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLCM.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 INSERTNO PIC S9(9) BINARY.
01 CTOK PIC X(12).
01 FBCODE.
   02 Condition-Token-Value.
       COPY CEEIGZCT.
       03 Case-1-Condition-ID.
           04 Severity PIC S9(4) BINARY.
           04 Msg-No PIC S9(4) BINARY.
       03 Case-2-Condition-ID
           REDEFINES Case-1-Condition-ID.
           04 Class-Code PIC S9(4) BINARY.
           04 Cause-Code PIC S9(4) BINARY.
       03 Case-Sev-Ctl PIC X.
       03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.
01 MSGDEST PIC S9(9) BINARY.
01 SEV PIC S9(4) BINARY.
01 MSGNO PIC S9(4) BINARY.
01 CASE PIC S9(4) BINARY.
01 SEV2 PIC S9(4) BINARY.
01 CNTRL PIC S9(4) BINARY.
01 FACID PIC X(3).
01 ISINFO PIC S9(9) BINARY.
01 VSTRING.
05 INSERT-TXTL  PIC S9(4) BINARY.
05 INSERT-TXT  PIC X(80).
PROCEDURE DIVISION.
PARA-CEPGCMI.

***********************************************************************
* Set up token fields for creation of a
* condition token for the user defined
* message file and message number.
***********************************************************************
MOVE 0 TO SEV.
MOVE 10 TO MSGNO.
MOVE 1 TO CASE.
MOVE 0 TO SEV2.
MOVE 0 TO CNTRL.
MOVE "XMP" TO FACID.
MOVE 0 TO ISINFO.

***********************************************************************
* Call CEENCOD to construct a condition token *
***********************************************************************
CALL "CEENCOD" USING SEV, MSGNO, CASE,
    SEV2, CNTRL, FACID,
    ISINFO, CTOK, FBCODE.

IF NOT CEE000 of FBCODE THEN
    DISPLAY "CEENCOD failed with msg"
    Msg-No of FBCODE UPON CONSOLE
    STOP RUN
END-IF.

***********************************************************************
* Call CEECMI to store and load message *
* insert 1. *
***********************************************************************
MOVE "<CEPGCMI"s insert>" TO INSERT-TXT.
MOVE 19 TO INSERT-TXTL.
MOVE 1 TO INSERTNO.
CALL "CEECMI" USING CTOK, INSERTNO, VSTRING.

***********************************************************************
* Call CEEMSG to write message to MSGFILE *
***********************************************************************
MOVE 2 TO MSGDEST.
CALL "CEEMSG" USING CTOK, MSGDEST, FBCODE.

IF NOT CEE000 of FBCODE THEN
    DISPLAY "CEEMSG failed with msg"
    Msg-No of FBCODE UPON CONSOLE
    STOP RUN
END-IF.

GOBACK.

Figure 59. COBOL Example of CEECMI (Part 2 of 2)
**PROCESS MACRO;
IBMCMI: Proc Options(Main);

/*Module/File Name: IBMCMI */
/*****************************/
**
** FUNCTION : CEECMI - store and load message *
** insert data *
** : CEEMSG - retrieve, format, and *
** dispatch a message to *
** message file *
**
** This example illustrates the invocation of *
** LE/370 message services to store and load *
** message insert data. The resulting message *
** and insert are written to the MSGFILE. *
**
****************************************************************************/
INCLUDE CEEIBMAW;
INCLUDE CEEIBMCT;
DECLARE INSERT CHAR(255) VARYING;
DCL 01 CTOK, /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
        05 Case BIT(2),
        05 Severity BIT(3),
        05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
    03 ISI /* Instance-Specific Information */
        REAL FIXED BINARY(31,0);
DCL 01 FBCODE, /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
        05 Case BIT(2),
        05 Severity BIT(3),
        05 Control BIT(3),
    03 FacID CHAR(3), /* Facility ID */
    03 ISI /* Instance-Specific Information */
        REAL FIXED BINARY(31,0);
DECLARE MSGFILE REAL FIXED BINARY(31,0);
DECLARE INSERT_NO REAL FIXED BINARY(31,0);

Figure 60. PL/I Example (Part 1 of 2)
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 CCEEDSHP (discard heap).

The number of heaps supported by Language Environment is limited only by the amount of virtual storage available.

Syntax

```
CEECRHP(heap_id, initial_size, increment, options, fc)
```

heap_id (output)

A fullword binary signed integer. heap_id is the heap identifier of the created heap. If a new heap cannot be created, the value of heap_id remains undefined.

Storage obtained from heap_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.

Figure 60. PL/I Example (Part 2 of 2)

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 CCEEDSHP (discard heap).

The number of heaps supported by Language Environment is limited only by the amount of virtual storage available.

Syntax

```
CEECRHP(heap_id, initial_size, increment, options, fc)
```

heap_id (output)

A fullword binary signed integer. heap_id is the heap identifier of the created heap. If a new heap cannot be created, the value of heap_id remains undefined.

Storage obtained from heap_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>
<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 <em>HEAP</em> run-time option)</td>
</tr>
<tr>
<td>01</td>
<td><em>HEAP(</em>,FREE) (location inherited from <em>HEAP</em> run-time option)</td>
</tr>
<tr>
<td>70</td>
<td><em>HEAP(</em>,KEEP) (location inherited from <em>HEAP</em> run-time option)</td>
</tr>
<tr>
<td>71</td>
<td><em>HEAP(</em>,ANYWHERE,KEEP)</td>
</tr>
<tr>
<td>72</td>
<td><em>HEAP(</em>,ANYWHERE,FREE)</td>
</tr>
<tr>
<td>73</td>
<td><em>HEAP(</em>,BELOW,KEEP)</td>
</tr>
<tr>
<td>74</td>
<td><em>HEAP(</em>,BELOW,FREE)</td>
</tr>
<tr>
<td>75</td>
<td><em>HEAP(</em>,ANYWHERE,) (disposition inherited from the <em>HEAP</em> run-time option)</td>
</tr>
<tr>
<td>76</td>
<td><em>HEAP(</em>,BELOW,) (disposition inherited from the <em>HEAP</em> run-time option)</td>
</tr>
<tr>
<td>77</td>
<td><em>HEAP(</em>,ANYWHERE,KEEP) (all heap storage obtained using this heap_id is allocated on a 4K boundary)</td>
</tr>
<tr>
<td>78</td>
<td><em>HEAP(</em>,ANYWHERE,FREE) (all heap storage obtained using this heap_id is allocated on a 4K boundary)</td>
</tr>
<tr>
<td>79</td>
<td><em>HEAP(</em>,ANYWHERE,KEEP) (all heap storage obtained using this heap_id is set to binary 0 when allocated using <em>CEEGETST</em>)</td>
</tr>
<tr>
<td>80</td>
<td><em>HEAP(</em>,ANYWHERE,FREE) (all heap storage obtained using this heap_id is set to binary 0 when allocated using <em>CEEGETST</em>)</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>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>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.

**For more information**

- See “CEEDSHP—Discard heap” on page 256 for more information about the CEEDSHP callable service.
- See “HEAP” on page 33 for more information about the HEAP run-time option and IBM-supplied defaults.
- See “HEAP” on page 33 for more information about the HEAP run-time option and IBM-supplied defaults.
/*Module/File Name: EDCCRHP */

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

int main(void) {

  _INT4 heapid, size, increment, options;
  _FEEDBACK fc;

  heapid = 0; /* heap identifier is set */
  /* by CECCRH */
  size = 4096; /* initial size of heap (in */
  /* bytes) */
  increment = 4096; /* increment to extend heap by */
  options = 72; /* set up heap as */
  /* (,,ANYWHERE,FREE) */

  /* create heap using CECCRH */
  CECCRH(heapid, size, increment, options, &fc);

  /* check the first 4 bytes of the feedback token */
  /* (0 if successful) */
  if (_FBCHECK(fc, CEE000) == 0) {
    printf("CECCRH failed with message number \%d\n", fc.fbk_msgno);
    exit(99);
  }

  /* discard the heap that was previously created */
  /* using CECCRH */
  CEEDSH(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.fbk_msgno);
    exit(99);
  }

  return 0;
}

Figure 61. C/C++ Example of CECCRHP
CBL LIB,QUOTE

*Module/File Name: IGZTCRHP

**********************************************
** Function: CEECRHP - create new additional **
** heap **
**
**********************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLCRHP.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HEAPID PIC S9(9) BINARY.
01 HPSIZE PIC S9(9) BINARY.
01 INCR PIC S9(9) BINARY.
01 OPTS PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.

**********************************************
** Specify 0 for HEAPID, and heap id will be **
** set by CEECRHP. **
** Heap size and increment will each be **
** 4096 bytes. **
** Specify 00 for OPTS, and HEAP attributes **
** will be inherited from the initial heap **
** (copied from the HEAP run-time option). **
**********************************************

MOVE 0 TO HEAPID.
MOVE 4096 TO HPSIZE.
MOVE 4096 TO INCR.
MOVE 00 TO OPTS.

CALL "CEECRHP" USING HEAPID, HPSIZE,
      INCR, OPTS, FC.
IF CEE000 of FC THEN
   DISPLAY "Created heap number " HEAPID
   " which is " HPSIZE " bytes long"
ELSE
   DISPLAY "CEECRHP failed with msg "
      Msg-No of FC UPON CONSOLE
   STOP RUN
END-IF.
GOBACK.

Figure 62. COBOL Example of CEECRHP
CEECZST—Reallocate (change size of) storage

CEECZST changes the size of a previously allocated heap element. The address parameter points to the beginning of the heap element. The new_size parameter gives the new size of the heap element, in bytes. The contents of the heap element are unchanged up to the shorter of the new and old sizes.
The CEECZST service returns a pointer to the reallocated heap element. It can move the storage location of the heap element. As a result, the `address` parameter passed to CEECZST is not necessarily the same as the value returned.

Because the new storage might be allocated at a different location from the existing allocation, any pointers (specifically any addresses) that referred to the old storage become invalid. Continued use of such dangling pointers gives unpredictable, and almost certainly incorrect, results.

The heap identifier is inferred from the address. The new storage block is allocated from the same heap that contained the old block.

The contents of the old storage are preserved in the following manner:
- If `new_size` is greater than the old size, the entire contents of the old storage block are copied to the new block. The remaining bytes in the new element are left uninitialized unless an initialization suboption value was specified for the heap in the STORAGE option.
- If `new_size` is less than the old size, the contents of the old block are truncated to the size of the new block.
- If `new_size` is equal to the old size, no operations are performed; a successful feedback code is returned.

Syntax

```plaintext
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.

For more information
- See "STORAGE" on page 69 for more information about the STORAGE run-time option.
- For information about CEEGTST, see “CEEGTST—Get heap storage” on page 320.
Examples

/*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 CEGTST */
  CEGTST(&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 CEGTST */
  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 64. C/C++ Example of CEECZST
CEECZST

CBL LIB,QUOTE
*Module/File Name: IGZTCZST
**************************************************
** Function: CEECZST - reallocate storage **
** **
** In this example, CEEGTST is called to **
** request storage from HEAPID = 0, and **
** CEECZST is called to change the size of **
** that storage request. **
** **
** **************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLCZST.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 HEAPID PIC S9(9) BINARY.
  01 HPSIZE PIC S9(9) BINARY.
  01 ADDRSS POINTER.
  01 NEWSIZE PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No  PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl  PIC X.
        03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
 PARA-CBLGTST.
**************************************************
** Specify 0 to get storage from the initial **
** heap. Specify 4000 to get 4000 bytes of **
** storage. **
**************************************************
MOVE 0 TO HEAPID.
MOVE 4000 TO HPSIZE.
**************************************************
** Call CEEGTST to obtain storage. **
**************************************************
CALL "CEEGTST" USING HEAPID, HPSIZE, ADDRSS, FC.

Figure 65. COBOL Example of CEECZST (Part 1 of 2)
If CEEGTST runs successfully, display result:

```
IF CEE000 OF FC THEN
  DISPLAY "HPSIZE"
  "bytes have been allocated."
ELSE
  DISPLAY "CEEGTST failed with msg"
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.
```

Specify a new size of 2000 bytes:

```
MOVE 2000 TO NEWSIZE.
```

Call CEECZST to change the size of the storage allocated in the call to CEEGTST:

```
CALL "CEECZST" USING ADDRSS, NEWSIZE, FC.
```

If CEECZST runs successfully, display result:

```
IF CEE000 OF FC THEN
  DISPLAY "The storage element now contains"
  "NEWSIZE" "bytes."
ELSE
  DISPLAY "CEEGTST failed with msg"
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.
GOBACK.
```

Figure 65. COBOL Example of CEECZST (Part 2 of 2)
*PROCESS MACRO;
/*Module/File Name: IBMZST        */
/*******************************
/** Function: CECEZST - reallocate storage  */
/**
/** In this example, CEETST is called to request 
/** storage from HEAPID = 0, and CECEZST is called**
/** to change the size of that storage request.  */
/**
/**
/*******************************
PLICZST: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL HEAPID REAL FIXED BINARY(31,0) ;
DCL STGSIZE REAL FIXED BINARY(31,0) ;
DCL ADDRSS1 POINTER;
DCL 01 FC1, /* Feedback token */
             03 MsgSev REAL FIXED BINARY(15,0),
             03 MsgNo REAL FIXED BINARY(15,0),
             03 Flags,
                 05 Case BIT(2),
                 05 Severity BIT(3),
                 05 Control BIT(3),
             03 FacID CHAR(3), /* Facility ID */
             03 ISI /* Instance-Specific Information */
               REAL FIXED BINARY(31,0);

Figure 66. PL/I Example of CECEZST (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

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.

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 29 on page 519 for a list of valid picture characters, and Table 30 on page 520 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 30 on page 520 for an additional example. Also see Table 31 on page 520 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 30 on page 520 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 234 for sample output dates. If input_Lilian_date is invalid, 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 invalid picture string was specified.</td>
</tr>
<tr>
<td>CEE2EQ</td>
<td>3</td>
<td>2522</td>
<td>&lt;JJJJ&gt;, &lt;CCCC&gt; or &lt;CCCCCCCC&gt; 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 an invalid DBCS string. 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 Programming Guide](https://www.ibm.com) or [COBOL for OS/390 & VM Programming Guide](https://www.ibm.com) 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 242 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 267 for information about how to get the default format for a given country code.
Examples

```c
/*Module/File Name: EDCDATE */
#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc;
    _INT4 lil_date = 139370; /* May 14, 1964 */
    _VSTRING date_pic,date;
    _CHAR80 date_out;
    strcpy(date_pic.string,
           "The date is Wwwwwwwzz, Mmmmmmmmnzz ZZ, YYYY");
    date_pic.length = strlen(date_pic.string);
    CEEDATE(&lil_date,&date_pic,date_out,&fc);
    if (_FBCHECK (fc , CEE000 ) != 0 ) {
        printf("CEEDATE failed with message number %d\n",fc.tok_msgno);
        exit(2999);
    }
    printf("%.80s\n",date_out);
}
```

Figure 67. C/C++ Example of CEEDATE

CBL LIB,QUOTE

```cbl
*Module/File Name: IGZDATE
******************************************************************************
**     Function: CEEDATE - convert Lilian date to **
**           character format **
**     In this example, a call is made to CEEDATE **
**     to convert a Lilian date (the number of **
**     days since 14 October 1582) to a character **
**     format (such as 6/22/88). The result is **
**     displayed. The Lilian date is obtained **
**     via a call to CEEDAYS. **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDATE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LILIAN PIC S9(9) BINARY.
01 CHRDATE PIC X(88).
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 68. COBOL Example of CEEDATE (Part 1 of 3)
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 CEE DAYS 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 CEDAYS 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 Mmmmmmmmmmmmz 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 68. COBOL Example of CEEDATE (Part 2 of 3)
CEEDATE

**************************************************************************
** If CEEDATE runs successfully, display result**
**************************************************************************

IF CEE000 of FC THEN
  DISPLAY "Input Lilian date of " LILIAN
  " corresponds to: " CHRDATE
ELSE
  DISPLAY "CEEDATE failed with msg "
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

GOBACK.

Figure 68. 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 CEEIBMAN;
%INCLUDE CEEIBMCT;

DCL LILIAN REAL FIXED BINARY(31,0);
DCL PICSTR CHAR(255) VARYING;
DCL CHRDATE CHAR(80);
DCL 01 FC, /* Feedback token */
03 MsgSev REAL FIXED BINARY(15,0),
03 MsgNo REAL FIXED BINARY(15,0),
03 Flags,
05 Case BIT(2),
05 Severity BIT(3),
05 Control BIT(3),
03 FacID CHAR(3), /* Facility ID */
03 ISI * Instance-Specific Information */
REAL FIXED BINARY(31,0);

LILIAN = 152385; /* input date in Lilian format */
/* picture string that describes how converted */
/* date is to be formatted */
PICSTR = 'ZD Mmmmmmmmmmmmmmz YYYY';

/* Call CEE3DATE to convert input Lilian date to */
/* a date in the character format specified in */
/* PICSTR */
CALL CEEDATE (LILIAN, PICSTR, CHRDATE, FC);

/* Print results if call to CEEDATE succeeds */
IF FBCHECK( FC, CEE000) THEN DO;
  PUT SKIP LIST( 'Lilian day ' || LILIAN
  || ' is equivalent to ' || CHRDATE );
END;
ELSE DO;
  DISPLAY( 'CEEDATE failed with msg'
    || FC.MsgNo );
  STOP;
END;

END PLIDATE;

Figure 69. 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>YYYYY</td>
<td>8805</td>
</tr>
<tr>
<td></td>
<td>YY-YY-MM</td>
<td>88-05</td>
</tr>
<tr>
<td></td>
<td>YYYY-MM-DD</td>
<td>880516</td>
</tr>
<tr>
<td></td>
<td>YYYYWMMDD</td>
<td>19880516</td>
</tr>
<tr>
<td></td>
<td>YYYY-MM-DD</td>
<td>1988-05-16</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>MMDDD</td>
<td>0517</td>
</tr>
<tr>
<td></td>
<td>MMM/DD</td>
<td>05/17</td>
</tr>
<tr>
<td></td>
<td>MMM/MDDYYYY</td>
<td>05/17/1988</td>
</tr>
<tr>
<td></td>
<td>MM/DD/YYYY</td>
<td>5/17/1988</td>
</tr>
<tr>
<td>148140</td>
<td>DD</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>DDMMD</td>
<td>1805</td>
</tr>
<tr>
<td></td>
<td>DDMMD/YYYY</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</td>
<td>88/05/20</td>
</tr>
<tr>
<td></td>
<td>HH:MI:SS.99</td>
<td>00:00:00.00</td>
</tr>
<tr>
<td></td>
<td>YYYY/ZM/ZD</td>
<td>1988/5/20 0:00 AM</td>
</tr>
<tr>
<td></td>
<td>ZZ:MI AP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YYYY/ZM/ZD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ZZ:MI AP</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="http://WWW">WWW</a>.. MMM DD, YYYY</td>
<td>SAT., MAY 21, 1988</td>
</tr>
<tr>
<td></td>
<td><a href="http://Www">Www</a>.., Mmm DD, YYYY</td>
<td>Sat., May 21, 1988</td>
</tr>
<tr>
<td></td>
<td>Wwwwwwwww</td>
<td>Saturday, May 21, 1988</td>
</tr>
<tr>
<td></td>
<td>Mmmmmmmmmmm</td>
<td>Saturday, May 21, 1988</td>
</tr>
<tr>
<td></td>
<td>DD, YYYY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WWWwwwwwwz, Mmmmmmmmmzz</td>
<td>Saturday, May 21, 1988</td>
</tr>
<tr>
<td></td>
<td>DD, YYYY</td>
<td></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.
Syntax

```plaintext
 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 29 on page 519 for a list of valid picture character terms and Table 30 on page 520 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 30 on page 520 for an example. See Table 31 on page 520 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 30 on page 520 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 242 for sample output.

If `input_seconds` is invalid, `output_timestamp` is set to all blanks and CEEDATM terminates with a non-CEE000 symbolic feedback code.

**fc (output)**

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

The following 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>
CEEDATM

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message number</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE2E9</td>
<td>3</td>
<td>2505</td>
<td>The number-of-seconds value was not within the supported range.</td>
</tr>
<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 invalid for CEEFMDT. The default date and time picture string datetime-string was returned.</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.
- 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 435 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 270 for information about how to get a default timestamp for a given country code.
/*Module/File Name: EDCDATM */

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

int main(void) {
    _FLOAT8 seconds = 12904183403.0;
    _VSTRING date,date_pic;
    _CHAR80 out_date;
    _FEEDBACK fc;

    strcpy(date_pic.string,
            "MMMMMMMMMZ 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\n", fc.tok_msgno);
        exit(2999);
    }

    printf("%.80s\n",out_date);
}

Figure 70. C/C++ Example of CEEDATM
**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 71. COBOL Example of CEEDATM (Part 1 of 3)
01 PICSTR.
  02 Vstring-length PIC S9(4) BINARY.
  02 Vstring-text.
    03 Vstring-char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of PICSTR.
  01 TIMESTP PIC X(80).
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
PARA-CBLDATM.
**********************************************************************
** Call CEESECS to convert timestamp of 6/2/88**
** at 10:23:45 AM to Lilian representation**
**********************************************************************
MOVE 20 TO Vstring-length of IN-DATE.
MOVE "06/02/88 10:23:45 AM" TO Vstring-text of IN-DATE.
MOVE 20 TO Vstring-length of PICSTR.
MOVE "MM/DD/YY HH:MI:SS AP" TO Vstring-text of PICSTR.
CALL "CEESECS" USING IN-DATE, PICSTR, SECONDS, FC.

**********************************************************************
** If CEESECS runs successfully, display result**
**********************************************************************
IF CEE000 of FC THEN
  DISPLAY Vstring-text of IN-DATE
  " is Lilian second: " SECONDS
ELSE
  DISPLAY "CEESECS failed with msg "
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

**********************************************************************
** Specify desired format of the output.**
**********************************************************************
MOVE 35 TO Vstring-length OF PICSTR.
MOVE "ZD Mmmmmmmmmmmmmmz YYYY at HH:MI:SS" TO Vstring-text OF PICSTR.

Figure 71. COBOL Example of CEDATM (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
  STOP RUN
END-IF.
GOBACK.

Figure 71. COBOL Example of CEEDATM (Part 3 of 3)
*PROCESS MACRO;
/**Module/File Name: IBMDATM

/****************************************************/
/** Function: CEEDATM - Convert seconds to character timestamp **/
/** In this example, CEEDATM is called to convert the number of seconds since 00:00:00 14 October 1582 to the character format specified in PICSTR. **/
/****************************************************/

PLIDATM: PROC OPTIONS(MAIN);

%INCLUDE CEEBMMAW;
%INCLUDE CEEBMCT;

DCL SECONDS REAL FLOAT DECIMAL(16);
DCL PICSTR CHAR(255) VARYING;
DCL TIMESTP CHAR(80);
DCL 01 FC, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
   05 Case BIT(2),
   05 Severity BIT(3),
   05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

SECONDS = 13166064060; /* Input is Lilian seconds*/

PICSTR = 'ZD Mmmmmmmmmmmm YYYY'; /* Picture */
   /* string describing desired output format */

/* Call CEEDATM to convert Lilian seconds to */
   /* format specified in PICSTR */
 CALL CEEDATM (SECONDS,PICSTR,TIMESTP,FC);

/* If CEEDATM ran successfully, print result */
IF FBCHECK(FC,CEE000) THEN DO;
   PUT SKIP LIST('Input Lilian seconds correspond to ' || TIMESTP);
   END;
ELSE DO;
   DISPLAY('CEEDATM failed with msg ' || FC.MsgNo);
   STOP;
   END;

END PLIDATM;

Figure 72. PL/I Example of CEEDATM
Table 27 shows the 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:MI: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>YYMMDDHHMISS</td>
<td>880516190001</td>
</tr>
<tr>
<td></td>
<td>YY-MM-DD HH:MI:SS</td>
<td>88-05-16</td>
</tr>
<tr>
<td></td>
<td>YYYY-MM-DD HH:MI: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>
<tr>
<td>12,799,191,661.986</td>
<td>DD Mm 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>Wwwwzz ZZ/DD/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>Maybbb</td>
</tr>
<tr>
<td></td>
<td>Mmmmmmmm</td>
<td>May</td>
</tr>
<tr>
<td></td>
<td>Mmmmmmmmmz</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>WWW</td>
<td>Mondayyyyyy</td>
</tr>
<tr>
<td></td>
<td>Wwww</td>
<td>Monday</td>
</tr>
<tr>
<td></td>
<td>Wwwwzz</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.

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).

CEEDAYS can convert a date into Lilian format. The inverse of CEEDAYS is CEEDATE, which converts output_Lilian_date from Lilian format to character format.
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 CEESCCEN.

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 29 on page 519 for a list of valid picture character terms that can be specified in `input_char_date`.

`picture_string (input)`

A halfword length-prefixed character string (VSTRING), indicating the format of the date specified in `input_char_date`.

Each character in the `picture_string` corresponds to a character in `input_char_date`. For example, if you specify MMDDYY as the `picture_string`, CEEDAYS reads an `input_char_date` of 060288 as 02 June 1988.

If delimiters such as a slash (/) appear in the `picture_string`, they count as placeholders but are otherwise ignored. See Table 29 on page 519 for a list of valid picture character terms and Table 30 on page 520 for examples of valid picture strings.

Whenever characters such as colons or slashes are included in the `picture_string` (such as HH:MM:SS YY/MM/DD), they count as placeholders but are otherwise ignored.
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 30 on page 520 for an additional example. See also Table 31 on page 520 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 30 on page 520 for an additional example.

**output_Lilian_date** (output)

A 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582. For example, 16 May 1988 is day number 148138.

If *input_char_date* does not contain a valid date, *output_Lilian_date* is set to 0 and CEEDAYS terminates with a non-CEE000 symbolic feedback code.

Date calculations are performed easily on the *output_Lilian_date*, because it is an integer. Leap year and end-of-year anomalies do not affect the calculations.

**fc** (output)

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

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 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 invalid 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 an invalid DBCS string. 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 Programming Guide` 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 227](#) for more information about the CEEDATE run-time option.
- See ["CEESCEN—Set the century window" on page 421](#) for more information about the CEESCEN callable service.

**Examples**

```c
/*Module/File Name: EDCDAYS */
#include <leawi.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc;
    _INT4 lil_date1,lil_date2;
    _VSTRING date,date_pic;

    /* use CEEDAYS to get the Lilian format */
    strcpy(date.string,"05/14/64");
    date.length = strlen(date.string);
    strcpy(date_pic.string,"MM/DD/YY");
    date_pic.length = strlen(date_pic.string);

    CEEDAYS(&date,&date_pic,&lil_date1,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) { 
        printf("CEEDAYS failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }

    /* use CEEDAYS to get the Lilian format */
    strcpy(date.string,"August 14, 1966");
    date.length = strlen(date.string);
    strcpy(date_pic.string,"Mmmmmmmm DD, YYYY");
    date_pic.length = strlen(date_pic.string);

    CEEDAYS(&date,&date_pic,&lil_date2,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) { 
        printf("CEEDAYS failed with message number %d\n", fc.tok_msgno);
        exit(2999);
    }

    /* subtract the two Lilian dates to find out */
    /* difference in days */
    printf("The number of days between"
    " May 14, 1964 and August 14, 1966"
    " is: %d\n",lil_date2 - lil_date1);
}
```

*Figure 73. C/C++ Example of CEEDAYS*
**Module/File Name:** IGZTDAYS

*******************************************
** Function: CEE DAYS - convert date to **
** Lilian format **
**
*******************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDAYS.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 CHRDATE.
   02 Vstring-length PIC 9(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 9(4) BINARY.
   02 Vstring-text.
      03 Vstring-char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of PICSTR.

01 LILIAN PIC 9(9) BINARY.
01 FC.
   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.
PARA-CBLDAYS.
*******************************************
** Specify input date and length **
*******************************************

MOVE 16 TO Vstring-length of CHRDATE.
MOVE "1 January 2000" TO Vstring-text of CHRDATE.

*Figure 74. COBOL Example of CEE DAYS (Part 1 of 2)*
** Specify a picture string that describes **
** input date, and the picture string's length.**

MOVE 25 TO Vstring-length of PICSTR.
MOVE "ZD MMMMMYYYY" 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 74. COBOL Example of CEEDAYS (Part 2 of 2)
CEEDAYS

*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 CEEIBMMAW;
%INCLUDE CEEIBMCT;

DCL CHRDATE CHAR(255) VARYING;
DCL CHRD2 CHAR(255) VARYING;
DCL PICSTR CHAR(255) VARYING;
DCL PICST2 CHAR(255) VARYING;
DCL LILIAN REAL FIXED BINARY(31,0);
DCL LIL2 REAL FIXED BINARY(31,0);
DCL 01 FC, /* Feedback token */
03 MsgSev REAL FIXED BINARY(15,0),
03 MsgNo REAL FIXED BINARY(15,0),
03 Flags, 
05 Case BIT(2),
05 Severity BIT(3),
05 Control BIT(3),
03 FacID CHAR(3), /* Facility ID */
03 ISI /* Instance-Specific Information */
   REAL FIXED BINARY(31,0);

/* First date to be converted to Lilian format */

Figure 75. PL/I Example of CEEDAYS (Part 1 of 2)
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 178 on page 452 for examples of how to alter the condition token field.

**Syntax**

\[
\text{CEEDCOD}(-\text{cond\_token}, -c_1, -c_2, -\text{case}, -\text{severity}, \ldots)
\]

\[
\text{CEEDCOD}(-\text{control}, -\text{facility\_ID}, -\text{i\_s\_info}, -\text{fc})
\]
**cond_token (input)**
A 12-byte condition token representing the current condition or feedback information.

**c_1 (output)**
A 2-byte binary integer representing the value of the first 2 bytes of the condition_ID.

**c_2 (output)**
A 2-byte binary integer representing the value of the second 2 bytes of the condition_ID. See [“CEENCOD—Construct a condition token” on page 400](#) 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 153 on page 402.

- z/OS UNIX consideration—In multithread applications, CEEDCOD affects only the calling thread.

For more information

- See the CEESGL HLL examples starting in Figure 178 on page 452 for examples of how to alter the condition token field.

- See “CEENCOD—Construct a condition token” on page 400 for a detailed explanation of the condition_ID.

- See CEENCOD Usage notes” on page 402 for a discussion of case 1 and case 2 types.

- See the facility_ID parameter of “CEENCOD—Construct a condition token” on page 400 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 449.
Examples

/*Module/File Name: EDCDCOD */

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

/**********************************************************/
/* In C/C++ it is not necessary to use this service.*/
/* The fields can be manipulated directly. See the */
/* example for CEESGL to see how to manipulate */
/* condition token fields directly. */
/**********************************************************/

int main(void) {
  _FEEDBACK fc,newfc;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi, heapid, size;
  _POINTER address;

  heapid = 0;
  size = 4000;

  CEEGTST(&heapid,&size,&address,&fc);
  if (!_FBCHECK (fc , CEE000 ) )
    {
      printf("CEEGTST failed with msgno %d\n", fc.tok_msgno);
      exit(2999);
    }

  /* decompose the feedback token to check for errors */
  CEECOD(&fc,&c_1,&c_2,&cond_case,&sev,&control,facid,; &isi,&newfc);

  if (!_FBCHECK (newfc , CEE000 ) )
    {
      printf("CEEDCOD failed with msgno %d\n", newfc.tok_msgno);
      exit(2889);
    }

  if (c_1 != 0 || c_2 != 0)
    printf("c_1 and c_2 returned from CEEDCOD should be 0\n");
  /* .
   . */
}

Figure 76. C/C++ Example of CEEDCOD
Figure 77. COBOL Example of CEEDCOD (Part 1 of 2)
PROCEDURE DIVISION.
****************************************************
** Call any Lang Env svc to receive a condition**
** token to use as input to CEEDCOD.  **
****************************************************

PARA-CBLGTST.
****************************************************
** Specify 0 to get storage from the initial  **
** heap. **
** Specify 4000 to get 4000 bytes of storage. **
** Call CEEGTST to obtain storage. **
****************************************************

MOVE 0 TO HEAPID.
MOVE 4000 TO HPSIZE.

CALL "CEEGTST" USING HEAPID, HPSIZE, 
ADDRSS, FC.

PARA-CBLDCOD.
****************************************************
** Use the FC returned from CEEGTST as an **
** input condition token to CEEDCOD. **
****************************************************

CALL "CEEDCOD" USING FC, SEV, MSGNO, CASE, 
SEV2, CNTRL, FACID, ISINFO, FC2.

IF CEE000 of FC2 THEN
DISPLAY "CEEGTST completed with msg " 
MSGNO ", Severity " SEV ", Case " 
CASE ", Control " CNTRL ", and " 
"Instance-Specific Information of " 
ISINFO "."
ELSE
DISPLAY "CEEDCOD failed with msg " 
Msg-No of FC2 UPON CONSOLE
END-IF.

GOBACK.

Figure 77. COBOL Example of CEEDCOD (Part 2 of 2)
CEEDCOD

*PROCESS MACRO;
/*Module/File Name: IBMDCOD
/************************************************************************/
/** **
/** Function: CEEDCOD - decompose a condition **/ **
/** token **/ **
/** **
/** In this example, a call is made to CEEGTST to **/ **
/** receive a condition token to decompose. **/ **
/** A call could have been made to any LE/370 **/ **
/** service. The condition token returned by **/ **
/** CEEGTST is used as input to CEEDCOD. **/ **
/** **
/**************************************************************************/
PLIDCOD: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL HEAPID REAL FIXED BINARY(31,0);
DCL STGSIZE REAL FIXED BINARY(31,0);
DCL ADDRESS 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);

Figure 78. PL/I Example of CEEDCOD (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 CEEDSHP call. A heap_id of 0 is not permitted with CEEDSHP.

Discarding a heap with CEEDSHP immediately returns all storage allocated to the heap to the operating system, even if the KEEP suboption has been specified with the HEAP run-time option.

```
DECLARE SEV REAL FIXED BINARY(15,0);
DECLARE MSGNO REAL FIXED BINARY(15,0);
DECLARE CASE REAL FIXED BINARY(15,0);
DECLARE SEV2 REAL FIXED BINARY(15,0);
DECLARE CNTRL REAL FIXED BINARY(15,0);
DECLARE FACID CHARACTER(3);
DECLARE ISINFO 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, FC);
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;
```

Figure 78. PL/I Example of CEEDCOD (Part 2 of 2)
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 invalid; the initial heap is logically created during enclave initialization and cannot be discarded.

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

The following 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>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 215.
- For more information about the CEEGTST callable service, and "CEEGTST—Get heap storage" on page 320.
Examples

```c
/*Module/File Name: EDCDSHP */

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

int main(void)
{
    _INT4 heapid, size, increment, options;
    _FEEDBACK fc;
    /* . */
    heapid = 0;    /* heap identifier is set */
        /* by CEECRHP */
    size = 4096;    /* initial size of heap */
        /* (in bytes) */
    increment = 4096;    /* increment to extend */
        /* the heap by */
    options = 72;    /* set up heap as */
        /* (,,ANYWHERE,FREE)*/

/* create heap using CEECRHP */
    CEECRHP(&heapid,&size,&increment,&options,&fc);

/* check the first 4 bytes of the feedback token */
/* (0 if successful) */
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEECRHP failed with message number %d\n", fc.tok_msgno);
        exit(99);
    }
    /* . */
    /* discard the heap that was previously created */
    /* using CEECRHP */
    CEEDSHP(&heapid,&fc);

/* check the first 4 bytes of the feedback token */
/* (0 if successful) */
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEDSHP failed with message number %d\n", fc.tok_msgno);
        exit(99);
    }
    /* . */
}
```

Figure 79. C/C++ Example of CEEDSHP
**Module/File Name: IGZTDSHP**

---

**Function: CEEDSHP - discard heap**

**In this example, a new additional heap is created a call to CEECRHP, and then discarded through a call to CEEDSHP.**

---

**IDENTIFICATION DIVISION.**

**PROGRAM-ID. CBLDSHP.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

```
01 HEAPID    PIC S9(9) BINARY.
01 HPSIZE    PIC S9(9) BINARY.
01 INCR      PIC S9(9) BINARY.
01 OPTS      PIC S9(9) BINARY.
01 FC.
  02 Condition-Token-Value.
COPY CEEIGZCT.
    03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No   PIC S9(4) BINARY.
    03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
    03 Case-Sev-Ctl PIC X.
    03 Facility-ID PIC XXX.
  02 I-S-Info  PIC S9(9) BINARY.
```

**PROCEDURE DIVISION.**

**PARA-CBLCRHP.**

---

**Specify 0 for HEAPID, and heap id will be set by CEECRHP.**

**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 "CEEECRHP" USING HEAPID, HPSIZE, INCR, OPTS, FC.
IF CEE000 of FC THEN
   DISPLAY "Created heap number " HEAPID " which is " HPSIZE " bytes long"
ELSE
   DISPLAY "CEECRHP failed with msg "
   Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.
```

---

Figure 80. 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 80. COBOL Example of CEEDSHP (Part 2 of 2)

*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 81. 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

```markdown
CEEDYWK(input_Lilian_date, output_day_no, fc)
```

### input_Lilian_date (input)

A 32-bit binary integer representing the Lilian date, the number of days since 14 October 1582.

For example, 16 May 1988 is day number 148138. The valid range of `input_Lilian_date` is between 1 and 3,074,324 (15 October 1582 and 31 December 9999).

### output_day_no (output)

A 32-bit binary integer representing `input_Lilian_date`'s day-of-week: 1 equals Sunday, 2 equals Monday, ..., 7 equals Saturday.

If `input_Lilian_date` is invalid, `output_day_no` is set to 0 and CEEDYWK terminates with a non-CEE000 symbolic feedback code.

### fc (output)

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

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

#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 82. C/C++ Example of CEEDYWK
CBL LIB,QUOTE

*Module/File Name: IGZTDYWK
**************************************************
**
** CBLDYWK - Call CEEDYWK to calculate the **
** day of the week from Lilian date **
**
** In this example, a call is made to CEEDYWK **
** to return the day of the week on which a **
** Lilian date falls. (A Lilian date is the **
** number of days since 14 October 1582) **
**
**************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDYWK.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 LILIAN PIC S9(9) BINARY.
  01 DAYNUM PIC S9(9) BINARY.
  01 IN-DATE.
    02 Vstring-length PIC S9(4) BINARY.
    02 Vstring-text.
      03 Vstring-char PIC X,
          OCCURS 0 TO 256 TIMES
          DEPENDING ON Vstring-length
          of IN-DATE.
    01 PICSTR.
      02 Vstring-length PIC S9(4) BINARY.
      02 Vstring-text.
      03 Vstring-char PIC X,
          OCCURS 0 TO 256 TIMES
          DEPENDING ON Vstring-length
          of PICSTR.
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
PARA-CBLDAYS.
** Call CEEDAYS to convert date of 6/2/88 to **
** Lilian representation **
  MOVE 6 TO Vstring-length of IN-DATE.
  MOVE "6/2/88" TO Vstring-text of IN-DATE(1:6).
  MOVE 8 TO Vstring-length of PICSTR.
  MOVE "MM/DD/YY" TO Vstring-text of PICSTR(1:8).
  CALL "CEEDAYS" USING IN-DATE, PICSTR, LILIAN, FC.

Figure 83. COBOL Example of CEEDYWK (Part 1 of 2)
** If CEEDAYS runs successfully, display result.
  IF CEE000 of FC THEN
    DISPLAY Vstring-text of IN-DATE
      " is Lilian day: " LILIAN
  ELSE
    DISPLAY "CEEDAYS failed with msg "
    Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

PARA-CBLDYW.

** 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.

Figure 83. COBOL Example of CEEDYWK (Part 2 of 2)
*PROCESS MACRO;
/*Module/File Name: IBMDYWK */
******************************************************************************
/** Function: CEEDYWK - calculate day of week **/
/** from Lilian date **/
/** In this example, CEEDYWK is called to **/
/** calculate the day of week on which a **/
/** Lilian date falls. **/
******************************************************************************

PLIDYWK: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL LILIAN 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;

Figure 84. PL/I Example of CEEDYWK
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 28 on page 515.

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 an invalid country_code, the default date format is 'YYYY-MM-DD'.

date_pic_str (output)
A fixed-length 80-character string (VSTRING), returned to the calling routine. It contains the default date picture string for the country specified. The picture string is left-justified and padded on the right with blanks if necessary.

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

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 notes

- z/OS UNIX considerations—CEEFMDA applies to the enclave. Every enclave has a single current country setting that has a single date format. Every thread in every enclave has the same default.

For more information

- For a list of the default settings for a specified country, see Table 28 on page 515.
- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 120 for an explanation of the CEE3CTY callable service.
Examples

/*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 85. C/C++ Example of CEEFMDA
CBL LIB,QUOTE
*Module/File Name: IGZTFMDA
****************************************************
** **
** CBLFMDA - Call CEEFMDA to obtain the **
** default date format **
****************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLFMDA.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 COUNTRY PIC X(2).
  01 PICSTR PIC X(80).
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLFMDA.
** Specify country code for the US and call **
** CEEFMDA to return the default date format **
for the US.
  MOVE "US" TO COUNTRY.
  CALL "CEEFMDA" USING COUNTRY, PICSTR, FC.

** If CEEFMDA runs successfully, display result **
IF CEE000 of FC THEN
  DISPLAY "The default date format for "
  "the US is: " PICSTR
ELSE
  DISPLAY "CEEFMDA failed with msg "
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.
GOBACK.

Figure 86. COBOL Example of CEEFMDA
CEEFMDT

CEEFMDT—Get default date and time format

CEEFMDT returns the default date and time picture strings for the country specified by *country_code*.

CEEFMDT is affected only by the country code setting of the COUNTRY run-time option or CEE3CTY callable service, not the CEESETL callable service or the setlocale() function.
Syntax

```c++
>>> CEEFMDT(_, country_code, datetime_str, fc _)
```

country_code (input)
A 2-character fixed-length string representing one of the country codes found in Table 28 on page 515.
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 an invalid country_code, the default date/time picture string is 'YYYY-MM-DD HH:MI:SS'.

datetime_str (output)
A fixed-length 80-character string (VSTRING), returned to the calling routine. It contains the default date and time picture string for the country specified. The picture string is left-justified and padded on the right with blanks, if necessary.

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

The following 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 country_code was invalid for CEEFMDT. The default date and time picture string datetime_str was returned.</td>
</tr>
</tbody>
</table>

Usage notes

- z/OS UNIX considerations—CEEFMDT applies to the enclave. Every enclave has a single current country setting that has a single date and time format. Every thread in every enclave has the same default.

For more information
- For a list of the default settings for a specified country, see Table 28 on page 515.
- See "COUNTRY" on page 22 for an explanation of the COUNTRY run-time option.
- See "CEE3CTY—Set default country" on page 120 for an explanation of CEE3CTY.
/*Module/File Name: EDCFMĐT */

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

int main(void) {
    _FEEDBACK fc;
    _CHAR2 country;
    _CHAR80 date_pic;

    /* get the default date and time format for Canada */
    memcpy(country,"CA",2);
    CEEFMĐT(country,date_pic,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEFMĐT failed with message number %d
", 
            fc.tok_msgno);
        exit(2999);
    }

    /* print out the default date and time format */
    printf("%.80s\n",date_pic);
}

Figure 88. C/C++ Example of CEEFMĐT
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 CEEE000 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 89. COBOL Example of CEEFMDT
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`.
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 105.

stringin (input)

An 8-byte field that contains the value of a double-precision floating point number.

maxsize (input)

A 4-byte integer that specifies the maximum number of bytes (including the string terminator) that can be placed in stringout.

format (input)

A halfword length-prefixed character string (VSTRING) of 256 bytes that contains plain characters and a conversion specification. Plain characters are copied to the output stream. Conversion specification results in the fetching of the input stringin argument that is converted and formatted.

A conversion specification consists of a percent character (%), optional flags, optional field width, optional left precision, optional right precision, and a required conversion character that determines the conversion to be performed.

Flags

You can specify one or more of the following flags to control the conversion.

= \f

An = followed by a single character f specifies that the character f should be used as the numeric fill character. The default numeric fill character is the space character. This option does not affect the other fill operations (such as field-width filling), which always use the space as the fill character.

^  

Do not format the currency amount with the grouping characters. The default is to insert the grouping characters if defined for the current locale.

+ or \( 

Specifies the style of representing positive and negative currency amounts. You must specify either + or -. If + is specified, the locale’s equivalent of + and - are used (for example, in USA: the empty (null) string if positive and - (minus sign) if negative). If - is specified, the locale’s equivalent of enclosing negative amounts within a parenthesis is used. If this option is not included, a default specified by the current locale is used.

!  

Suppresses the currency symbol from the output conversion.

Field Width

w  

The decimal digit string w specifies a minimum field width in which the result of the conversion is right-justified (or left-justified if -w is specified).
Left Precision

#n  A # (pound sign) followed by the decimal digit string \( n \) specifies a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to the CEEFMON service aligned in the same columns. It can also be used to fill unused positions with a special character as in $***123.45. This option causes an amount to be formatted as if it has the number of digits specified by \( n \). If more digit positions are required than are specified, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character. See the =f specification above.

If grouping is enabled, it is applied to the combined fill characters plus regular digits. However, if the fill character is not 0 (digit zero), grouping separators inserted after a fill character are replaced by the same fill character ($0,001,234.56 versus $****1,234.56).

Right Precision

.p  A period followed by the decimal digit string \( p \) specifies the number of digits after the radix character. If the value of the precision \( p \) is zero, no radix character appears. If this option is not included, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting.

Conversion Characters

i  The double argument is formatted according to the locale’s international currency format (for example, in USA: USD 1,234.56).

n  The double argument is formatted according to the locale’s national currency format (for example, in USA: $1,234.56).

%  No argument is converted; the conversion specification %% is replaced by a single %.

stringout (output)
Contains the output of the CEEFMON service.

result (output)
Contains the number of characters placed in stringout, if successful. If unsuccessful, an error is reported.

fc (output/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 120, and "CEE3LNG—Set national language" on page 163.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 443.

Examples

```cobol
CBL LIB,QUOTE
*Module/File Name: IGZTFMON
******************************************************************************
* Example for callable service CEEFMON  *
* Function: Convert a numeric value to a  *
* monetary string using specified  *
* format passed as parameter.  *
* Valid only for COBOL for MVS & VM Release 2  *
* or later.  *
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. COBFMON.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 Monetary              COMP-2.
  01 Max-Size PIC S9(9) BINARY.
  01 Format-Mon.
      02 FM-Length PIC S9(4) BINARY.
      02 FM-String PIC X(256).
  01 Output-Mon.
      02 OM-Length PIC S9(4) BINARY.
      02 OM-String PIC X(60).
  01 Length-Mon PIC S9(9) BINARY.
  01 Locale-Name.
      02 LN-Length PIC S9(4) BINARY.
      02 LN-String PIC X(256).
** Use Locale category constants
```

Figure 91. COBOL Example of CEEFMON (Part 1 of 2)
**COPY CEEIGZLC.**

**COPY CEEIGZCT.**

**PROCEDURE DIVISION.**
**Set up locale name for United States**
MOVE 14 TO LN-Length.
MOVE "En_US.IBM-1047" TO LN-String (1:LN-Length).
**Set all locale categories to United States.**
**Use LC-ALL category constant from CEEIGZLC.**
CALL "CEESETL" USING Locale-Name, LC-ALL, FC.
**Check feedback code**
IF Severity > 0
 DISPLAY "Call to CEESETL failed. " Msg-No
 STOP RUN
END-IF.
**Set up numeric value**
MOVE 12345.62 TO Monetary.
MOVE 60 TO Max-Size.
MOVE 2 TO FM-Length.
MOVE "%i" TO FM-String (1:FM-Length).
**Call CEEFMON to convert numeric value**
CALL "CEEFMON" USING OMITTED, Monetary, Max-Size, Format-Mon Output-Mon, Length-Mon, FC.
**Check feedback code and display result**
IF Severity > 0
 DISPLAY "Call to CEEFMON failed. " Msg-No
ELSE
 DISPLAY "International format is " OM-String(1:OM-Length)
END-IF.
STOP RUN.
END PROGRAM COBFMON.
*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 CEEIBMAN; /* 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;

Figure 92. PL/I Example of CEEFMON (Part 1 of 2)
CEEFMTM

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 28 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**

```
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 28 on page 515].

*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 an invalid *country_code*, the default time picture string is 'HH:MI:SS'.

**time_pic_str (output)**

A fixed-length 80-character string (VSTRING), representing the default time picture string for the country specified. The picture string is left-justified and padded on the right with blanks if necessary.
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>CEE3CD</td>
<td>2</td>
<td>3469</td>
<td>The country code <code>country_code</code> was invalid for CEEFMTM. The default time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>picture string <code>time_pic_string</code> was returned.</td>
</tr>
<tr>
<td>CEE3CE</td>
<td>2</td>
<td>3470</td>
<td>The date and time string <code>datetime_str</code> was truncated and was not defined in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CEEFMDT.</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 120 for an explanation of the CEE3CTY callable service.

Examples

```c
/*Module/File Name: EDFCFMTM */
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedcct.h>

int main(void) {
    _FEEDBACK fc;
    _CHAR2 country;
    _CHAR80 time_pic;

    /* get the default time format for Canada */
    memcpy(country,"CA",2);
    CEEFMTM(country,time_pic,&fc);
    if (_FBCHECK (fc, CEE000 ) != 0 ) {
        printf("CEEFMTM failed with message number %d\n",fc.tok_msgno);
        exit(2999);
    }
    /* print out the default time format */
    printf("%.80s\n",time_pic);
}
```

Figure 93. C/C++ Example of CEEFMTM
**IGZTFMTM** - Call CEEFMTM to obtain the default time format

**IDENTIFICATION DIVISION.**
**PROGRAM-ID.** IGZTFMTM.
**DATA DIVISION.**
**WORKING-STORAGE SECTION.**
**01** COUNTRY PIC X(2).
**01** PICSTR PIC X(80).
**01** FC.
**02** Condition-Token-Value.
COPY CEEI5ZCT.
**03** Case-1-Condition-ID.
**04** Severity PIC S9(4) BINARY.
**04** Msg-No PIC S9(4) BINARY.
**03** Case-2-Condition-ID
REDEFINES Case-1-Condition-ID.
**04** Class-Code PIC S9(4) BINARY.
**04** Cause-Code PIC S9(4) BINARY.
**03** Case-Sev-Ctl PIC S9(4) BINARY.
**03** Facility-ID PIC XXX.
**02** I-S-Info PIC S9(9) BINARY.
**PROCEDURE DIVISION.**
**PARA-CBLFMTM.**
**Specify country code for the US.**
MOVE "US" TO COUNTRY.
**Call CEEFMTM to return the default time format**
**for the US.**
CALL "CEEFMTM" USING COUNTRY, PICSTR, FC.
**If CEEFMTM runs successfully, display result.**
IF CEE000 of FC THEN
DISPLAY "The default time format for "
"the US is: " PICSTR
ELSE
DISPLAY "CEEFMTM failed with msg "
Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.
GOBACK.

---

Figure 94. COBOL Example of CEEFMTM
**CEEFRST**

**Free heap storage**

CEEFRST frees storage previously allocated by CEEGTST or by a language intrinsic function. Normally, you do not need to call CEEFRST because Language Environment automatically returns all heap storage to the operating system when the enclave terminates. However, if you are allocating a large amount of heap storage, you should free the storage when it is no longer needed. This freed storage then becomes available for later requests for heap storage, thus reducing the total amount of storage needed to run the application.
All requests for storage are conditional. If storage is not available, the feedback code (fc) is set and returned to you, but the thread does not abend. An attempt to free storage that was already marked as free produces no action and returns a non-CEE000 symbolic feedback code. An attempt to free storage at anything other than a valid starting address produces no action and returns a non-CEE000 symbolic feedback code. The application does not abend.

However, if you call CEEFRST for an invalid address, and you had specified TRAP(OFF), your application can abend. Language Environment’s reaction to this is undefined. Also, partial freeing of an allocated area is not supported.

When storage is allocated by CEEGTST, its allocated size is used during free operations. Storage allocated by CEEGTST, but not explicitly freed, is automatically freed at enclave termination.

CEEFRST generates a system-level free storage call to return a storage increment to the operating system only when:
- The last heap element within an increment is being freed, and
- The HEAP run-time option or a call to CEECRHP specifies FREE (note that KEEP is the IBM-supplied default setting for the initial heap).

Otherwise, the freed storage is simply added to the free list; it is not returned to the operating system until termination. The out-of-storage condition can cause freeing of empty increments even when KEEP is specified.

**Syntax**

```
CEEFRST(address, fc)
```

*address* (input)

A fullword address pointer. *address* is the address returned by a previous CEEGTST call or a language intrinsic function such as ALLOCATE or malloc(). The storage at this address is deallocated.

*fc* (output)

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

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 (CEEFRST) request was not recognized,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or heap storage (CEEGZST) 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 CEEGTST call. If the pointer is not reassigned, any further use of it causes unpredictable results.
- z/OS UNIX considerations—CEEFRST applies to the enclave. One thread can allocate storage, and another can free it.

For more information

- See "CEEGTST—Get heap storage" on page 320 for more information about the CEEGTST callable service.
- See "HEAP" on page 33 for further information about the HEAP run-time option.
- See "CEECRHP—Create new additional heap" on page 215 for more information about the CEECRHP callable service.
- See "STORAGE" on page 69 for further information about the STORAGE run-time option.
Examples

```c
#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 96. C/C++ Example of CEEFRST
CBL LIB,QUOTE
*Module/File Name: IGZTFRST
******************************************************************************
**
** IGZTFRST - Call CEEFRST to free heap  **
** storage  **
**
** In this example, a call is made to  **
** CEEGTST to obtain 4000 bytes of storage  **
** from the initial heap (HEAPID = 0).  **
** A call is then made to CEEFRST to free  **
** the storage.  **
**
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTFRST.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 HEAPID PIC S9(9) BINARY.
  01 STGSIZE PIC S9(9) BINARY.
  01 ADDRSS USAGE IS POINTER.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
          03 Case-2-Condition-ID
            REDEFINES Case-1-Condition-ID.
              04 Class-Code PIC S9(4) BINARY.
              04 Cause-Code PIC S9(4) BINARY.
              03 Case-Sev-Ctl PIC X.
              03 Facility-ID PIC XXX.
          02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLFRST.
** Specify 0 to get storage from the initial **
** heap.  **
** Specify 4000 to get 4000 bytes of storage. **
** Call CEEGTST to obtain storage. **
  MOVE 0 TO HEAPID.
  MOVE 4000 TO STGSIZE.
  CALL "CEEGTST" USING HEAPID, STGSIZE, ADDRSS, FC.
  IF CEE000 of FC THEN
    DISPLAY "Obtained " STGSIZE " bytes of"
    " storage at location " ADDRSS
    " from heap number " HEAPID
  ELSE
    DISPLAY "CEEGTST failed with msg "
    Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

Figure 97. COBOL Example of CEEFRST (Part 1 of 2)
** 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 97. COBOL Example of CEEFRST (Part 2 of 2)

*PROCESS MACRO;
/*Module/File Name: IBMFRST */
/***********************************/
/**
/** Function: CEEFRST - free heap storage **/
/**
/** This example calls CEEGTST to obtain storage **/
/** from the initial heap, and then calls **/
/** CEEFRST to discard it. **/
/**
/***********************************/
PLIFRST: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL ADDRSS  POINTER;
DCL HEAPID  REAL FIXED BINARY(31,0);
DCL STGSIZE REAL FIXED BINARY(31,0);
DCL 01 FC, /* Feedback token */
   03 MsgSev  REAL FIXED BINARY(15,0),
   03 MsgNo   REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case   BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID   CHAR(3), /* Facility ID */
   03 ISI    /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

DCL 01 FC2,  /* Feedback token */
   03 MsgSev  REAL FIXED BINARY(15,0),
   03 MsgNo   REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case   BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID   CHAR(3), /* Facility ID */
   03 ISI    /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

HEAPID = 0; /* get storage from the initial heap */

Figure 98. PL/I Example of CEEFRST (Part 1 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**

```
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 105.
**time_and_date (input)**

Points to the time structure that is to be converted.

**td_sec**

A 4-byte integer that describes the number of seconds in the range of 0 through 59.

**td_min**

A 4-byte integer that describes the number of minutes in the range of 0 through 59.

**td_hour**

A 4-byte integer that describes the number of hours in the range of 0 through 23.

**td_mday**

A 4-byte integer that describes the day of the month in the range of 1 through 31.

**td_mon**

A 4-byte integer that describes the month of the year in the range of 0 through 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.
- An optional decimal digit string specifies a minimum field width. A converted value that has fewer characters than the field width is padded with spaces to the left. If the decimal digit string is preceded by a minus sign, padding with spaces occurs to the right of the converted value.

If no width is given, for numeric fields the appropriate default width is used with the field padded on the left with zeros as required. For strings, the output field is made exactly wide enough to contain the string.
An optional precision value gives the maximum number of characters to be printed for the conversion specification. The precision value is a decimal digit string preceded by a period. If the value to be output is longer than the precision, it is truncated on the right.

The type of conversion is specified by one or two conversion characters. The characters and their meanings are:

- `%a` The abbreviated weekday name (for example, Sun) defined by the `abday` statement in the current locale.
- `%A` The full weekday name (for example, Sunday) defined by the `day` statement in the current locale.
- `%b` The abbreviated month name (for example, Jan) defined by the `abmon` statement in the current locale.
- `%B` The full month name (for example, January) defined by the `month` statement in the current locale.
- `%c` The date and time format defined by the `d_t_fmt` statement in the current locale.
- `%d` The day of the month as a decimal number (01 to 31).
- `%D` The date in %m/%d/%y format (for example, 01/31/91).
- `%e` The day of the month as a decimal number (1 to 31). The `%e` field descriptor uses a two-digit field. If the day of the month is not a two-digit number, the leading digit is filled with a space character.
- `%E` The combined alternative era year and name, respectively, in %o %N format in the current locale.
- `%h` The abbreviated month name (for example, Jan) defined by the `abmon` statement in the current locale. This field descriptor is a synonym for the `%b` field descriptor.
- `%H` The 24-hour clock hour as a decimal number (00 to 23).
- `%I` The 12-hour clock hour as a decimal number (01 to 12).
- `%j` The day of the year as a decimal number (001 to 366).
- `%m` The month of the year as a decimal number (01 to 12).
- `%M` The minutes of the hour as a decimal number (00 to 59).
- `%n` Specifies a new-line character.
- `%N` The alternative era name in the current locale.
- `%o` The alternative era year in the current locale.
- `%p` The AM or PM string defined by the `am_pm` statement in the current locale.
- `%r` The 12-hour clock time with AM/PM notation as defined by the `t_fmt_ampm` statement (%I:%M:%S [AM|PM]) in the current locale.
- `%S` The seconds of the minute as a decimal number (00 to 59).
- `%t` Specifies a tab character.
- `%T` Represents 24-hour clock time in the format %H:%M:%S (for example, 16:55:15).
- `%U` The week of the year as a decimal number (00 to 53). Sunday, or its
equivalent as defined by the *day* statement, is considered the first
day of the week for calculating the value of this field descriptor.

`%w` The day of the week as a decimal number (0 to 6). Sunday, or its
equivalent as defined by the *day* statement, is considered as 0 for
calculating the value of this field descriptor.

`%W` The week of the year as a decimal number (00 to 53). Monday, or its
equivalent as defined by the *day* statement, is considered the first
day of the week for calculating the value of this field descriptor.

`%x` The date format defined by the *d_fmt* statement in the current locale.

`%X` The time format defined by the *t_fmt* statement in the current locale.

`%y` The year of the century (00 to 99).

`%Y` The year as a decimal number (for example, 1989).

`%Z` The time-zone name, if one can be determined (for example, EST);
no characters are displayed if a time zone cannot be determined.

`%%` Specifies a percent sign (%) character.

**stringout (output)**

A halfword length-prefixed character string (VSTRING) of 256 bytes that
contains the formatted time and date output of the CEEFTDS service.

**fc (output/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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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 120, and "CEE3LNG—Set national
  language" on page 163.
- For more information about the CEESETL callable service, see "CEESETL—Set
  locale operating environment" on page 443.
Examples

CBL LIB,QUOTE
*Module/ File Name: IGZTFTDS
*****************************************************
* Example for callable service CEEFTDS *
* Function: Convert numeric time and date *
* values to a string using specified *
* format string and locale format *
* conversions. *
* Valid only for COBOL for MVS & VM Release 2 *
* or later. *
*****************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINFTDS.
DATA DIVISION.
WORKING-STORAGE SECTION.
* Use TD-Struct for CEEFTDS calls
COPY CEEIGZTD.
*
PROCEDURE DIVISION.
* Subroutine needed for pointer addressing
  CALL "COBFTDS" USING TD-Struct.
  STOP RUN.
*
IDENTIFICATION DIVISION.
PROGRAM-ID. COBFTDS.
DATA DIVISION.
WORKING- STORAGE SECTION.
* Use Locale category constants
COPY CEEIGZLC.
*
01 Ptr-FTDS POINTER.
01 Output-FTDS.
  02 O-Length PIC S9(4) BINARY.
  02 O-String PIC X(72).
01 Format-FTDS.
  02 F-Length PIC S9(4) BINARY.
  02 F-String PIC X(64).
01 Max-Size PIC S9(9) BINARY.

Figure 99. COBOL Example of CEEFTDS (Part 1 of 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-Ctl1 PIC X.
   03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.

LINKAGE SECTION.
* Use TD-Struct for calls to CEEFTDS
COPY CEEIGZTD.
* PROCEDURE DIVISION USING TD-Struct.
* Set up time and date values
   MOVE 1 TO TM-Sec.
   MOVE 2 TO TM-Min.
   MOVE 3 TO TM-Hour.
   MOVE 9 TO TM-Day.
   MOVE 11 TO TM-Mon.
   MOVE 94 TO TM-Year.
   MOVE 5 TO TM-Wday.
   MOVE 344 TO TM-Yday.
   MOVE 1 TO TM-Is-DLST.

* Set up format string for CEEFTDS call
   MOVE 72 TO Max-Size.
   MOVE 36 TO F-Length.
   MOVE "Today is %A, %b %d Time: %I:%M %p"
      TO F-String (1:F-Length).

* Set up pointer to structure for CEEFTDS call
   SET Ptr-FTDS TO ADDRESS OF TD-Struct.

* Call CEEFTDS to convert numeric values
   CALL "CEEFTDS" USING OMITTED, Ptr-FTDS,
      Max-Size, Format-FTDS, Output-FTDS, FC.

* Check feedback code and display result
   IF Severity = 0
      DISPLAY "Format " F-String (1:F-Length)
      DISPLAY "Result " O-String (1:O-Length)
   ELSE
      DISPLAY "Call to CEEFTDS failed. " Msg-No
   END-IF.

   EXIT PROGRAM.
END PROGRAM COBFTDS.
* END PROGRAM MAINFTDS.

Figure 99. COBOL Example of CEEFTDS (Part 2 of 2)
*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 */
    **************************************************************************/

    PLIFTD: 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 */

Figure 100. PL/I Example of CEEFTDS (Part 1 of 2)
CEEGMT—Get current Greenwich Mean Time

CEEGMT returns the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. The returned values are compatible with those generated and used by the other Language Environment date and time services.

In order for the results of this service to be meaningful, your system’s TOD (time-of-day) clock must be set to Greenwich Mean Time and be based on the standard epoch. Use CEEGMTO to get the offset from GMT to local time.

The values returned by CEEGMT are handy for elapsed time calculations. For example, you can calculate the time elapsed between two calls to CEEGMT by calculating the differences between the returned values.

Language Environment treats Coordinated Universal Time (UTC) and Greenwich Mean Time (GMT) as the same. You can use the CEEUTC service, which is an alias of the CEEGMT service, to get the same value.

Figure 100. PL/I Example of CEEFTDS (Part 2 of 2)
Syntax

```plaintext
>>> CEEGMT(output_GMT_Lilian, output_GMT_seconds, fc)
```

`output_GMT_Lilian (output)`

A 32-bit binary integer representing the current time in Greenwich, England, in the Lilian format (the number of days since 14 October 1582).

For example, 16 May 1988 is day number 148138. If GMT is not available from the system, `output_GMT_Lilian` is set to 0 and CEEGMT terminates with a non-CEE000 symbolic feedback code.

`output_GMT_seconds (output)`

A 64-bit double floating-point number representing the current date and time in Greenwich, England, as the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds.

For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). 19:00:01.078 on 16 May 1988 is second number 12,799,191,601.078. If GMT is not available from the system, `output_GMT_seconds` is set to 0 and CEEGMT terminates with a non-CEE000 symbolic feedback code.

`fc (output)`

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

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—CEEGMT does not use the OS TIME macro.
- z/OS UNIX consideration—In multithread applications, CEEGMT affects only the calling thread.
- Setting the TOD (time-of-day) clock to anything before January 1, 1972 may produce unpredictable results in your applications.

**For more information**

See “CEEGMTO—Get offset from Greenwich Mean Time to local time” on page 300 for more information about the CEEGMTO callable service.
Examples

/*Module/File Name: EDCGMT */

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

int main(void) {

_FEEDBACK fc;
_INT4 lilGMT_date;
_FLOAT8 secGMT_date;

CEEGMT(&lilGMT_date,&secGMT_date,&fc);
if (_FBCHECK (fc, CEE000) != 0 ) {
   printf("CEEGMT failed with message number %d\n",fc.tok_msgno);
   exit(2999);
}
printf("The current Lilian date in Greenwich,\nEngland is %d\n", lilGMT_date);
}

Figure 101. C/C++ Example of CEEGMT
CBL LIB,QUOTE
*Module/File Name: IGZTGMT
*****************************************************************************
** **
** IGZTGMT - Call CEEGMT to get current **
** Greenwich Mean Time **
** **
** In this example, a call is made to CEEGMT **
** to return the current GMT as a Lilian date **
** and as Lilian seconds. The results are **
** displayed. **
** **
*****************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGMT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LILIAN PIC S9(9) BINARY.
01 SECS COMP-2.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLGMT.
   CALL "CEEGMT" USING LILIAN , SECS , FC.

   IF CEE000 of FC THEN
      DISPLAY "The current GMT is also"
      "known as Lilian day: " LILIAN
      DISPLAY "The current GMT in Lilian"
      "seconds is: " SECS
   ELSE
      DISPLAY "CEEGMT failed with msg"
      Msg-No of FC UPON CONSOLE
      STOP RUN
   END-IF.
GOBACK.

Figure 102. COBOL Example of CEEGMT
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).
offset_hours (output)
A 32-bit binary integer representing the offset from GMT to local time, in hours.
For example, for Pacific Standard Time, offset_hours equals −8.
The range of offset_hours is −12 to +13 (+13 = Daylight Savings Time in the +12 time zone).
If local time offset is not available, offset_hours equals 0 and CEEGMTO terminates with a non-CEE000 symbolic feedback code.

offset_minutes (output)
A 32-bit binary integer representing the number of additional minutes that local time is ahead of or behind GMT.
The range of offset_minutes is 0 to 59.
If the local time offset is not available, offset_minutes equals 0 and CEEGMTO terminates with a non-CEE000 symbolic feedback code.

offset_seconds (output)
A 64-bit double floating-point number representing the offset from GMT to local time, in seconds.
For example, Pacific Standard Time is eight hours behind GMT. If local time is in the Pacific time zone during standard time, CEEGMTO would return −28,800 (−8 * 60 * 60). The range of offset_seconds is −43,200 to +46,800.
offset_seconds can be used with CEEGMT to calculate local date and time.
See “CEEGMT—Get current Greenwich Mean Time” on page 296 for more information.
If the local time offset is not available from the system, offset_seconds is set to 0 and CEEGMTO terminates with a non-CEE000 symbolic feedback code.

fc (output)
A 12-byte feedback code, optional in some languages, that indicates the result of this service.
The following 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—CEEGMTO does not use the OS TIME macro.
- z/OS UNIX consideration—In multithread applications, CEEGMTO affects only the calling thread.
For more information

- See "CEEDATM—Convert seconds to character timestamp" on page 234 for more information about the CEEDATM callable service.
Examples

/*@Module/File Name: EDCGMO  */
#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;
    CEEGMO(&GMT_hours,&GMT_mins,&GMT_secs,&fc);
    if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
        printf("CEEGMO 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 104. C/C++ Example of CEEGMO

CBL LIB,QUOTE
(Module/File Name: IGZTGMTO
************************************************************************************************
** **
** IGZTGMTO - Call CEEGMO to get offset from **
** Greenwich Mean Time to local **
** time **
** **
** In this example, a call is made to CEEGMO **
** to return the offset from GMT to local time **
** as separate binary integers representing **
** offset hours, minutes, and seconds. The **
** results are displayed. **
**
************************************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGMTO.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 HOURS PIC S9(9) BINARY.
01 MINUTES PIC S9(9) BINARY.
01 SECONDS COMP-2.
01 FC.
    02 Condition-Token-Value.
COPY CEEIGZCT.
    03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
    03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
    03 Case-Sev-Ctl PIC X.
    03 Facility-ID PIC XXX.
02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
CEEGMTO

*PROCESS MACRO;
/* Module/File Name: IBMGMTO */
******************************************************************************
/** **
/** Function: CEEGMTO - get the offset from **
/** Greenwich Mean Time **
/** to local time **
/** **
******************************************************************************

PLIGMTO: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL HOURS REAL FIXED BINARY(31,0);
DCL MINUTES REAL FIXED BINARY(31,0);
DCL SECONDS REAL FIXED BINARY(31,0);
DCL 01 FC, /* Feedback token */
  03 MsgSev REAL FIXED BINARY(15,0),
  03 MsgNo REAL FIXED BINARY(15,0),
  03 Flags,
    05 Case BIT(2),
    05 Severity BIT(3),
    05 Control BIT(3),
  03 FacID CHAR(3), /* Facility ID */
  03 ISI /* Instance-Specific Information */
    REAL FIXED BINARY(31,0);

/* Call CEEGMTO to return hours, minutes, and */
/* seconds that local time is offset from GMT */

CALL CEEGMTO ( HOURS, MINUTES, SECONDS, FC );

/* If CEEGMTO ran successfully, print results */
IF FBCHECK( FC, CEE000 ) THEN DO;
  PUT SKIP EDIT('The difference between GMT and '
|| 'local time is ', HOURS, ':', MINUTES )
    (A, P'S99', A, P'99' );
  END;
ELSE DO;
  DISPLAY( 'CEEGMTO failed with msg ' || FC.MsgNo );
  STOP;
  END;
END PLIGMTO;

Figure 106. PL/I Example of CEEGMTO

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

```c
-- CEEGPID(--CEE_Version_ID--,__Plat_ID__,__fc__)```

### CEE_Version_ID (output)

A fullword integer representing the version of Language Environment that created this data block. The current value of this parameter is still a fullword, but can be interpreted as a four-byte hex number,

<table>
<thead>
<tr>
<th>PP</th>
<th>VV</th>
<th>RR</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Number</td>
<td>Version</td>
<td>Release</td>
<td>Modification</td>
</tr>
</tbody>
</table>

### 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.
Examples

```c
/*Module/File Name: EDDCPID */
/****************************/
/*
/* Licensed Materials - Property of IBM */
/* */
/* 5647-A01 5688-198 */
/* */
/* (C) Copyright IBM Corp. 1991, 2000 */
/* */
/* US Government Users Restricted Rights - Use, */
/* duplication or disclosure restricted by GSA */
/* */
/* ADP Schedule Contract with IBM Corp. */
/* */
/* STATUS = HLE7703 */
/****************************/
/****************************/
/* Note that the format of data returned by CEEGPID */
/* changed in OS/390 V2R10. This sample tests the */
/* version and chooses the appropriate format. */
/****************************/
/****************************/
#include (stdio.h)
#include (string.h)
#include (stdlib.h)
#include (leawi.h)
#include (ceeedcct.h)

int main(void)
{
    _INT4 cee_:ver_:id,
    plat_id;
    _FEEDBACK fc;
    int Vmask= 0x00FF0000;
    int Rmask= 0x0000FF00;
    int Mmask= 0x000000FF;
    /* get the LE version and the platform id */
    CEEGPID(cee_ver_id,plat_id,fc);
    if (_FBCHECK (fc , CEE000 ) != 0 )
    {
        printf("CEEGPID failed with message number %d
",
            fc.tok_msgno);
        exit(2999);
    }
    /* If platform is z/OS and LE is at release 2.10 or later, */
    /* use the new interface definition. */
    if ((plat_id == 3) & (cee_ver_id > 290))
    {
        printf("The LE version id is %08X\n",cee_ver_id);
        printf("Version: %d\n",(Vmask & cee_ver_id)>>16);
        printf("Release: %d\n",(Rmask & cee_ver_id)>>8);
        printf("Modification: %d\n",Mmask & cee_ver_id);
    }
    else
    {
        printf("The current platform is \n",cee_ver_id);
        printf("The LE version is %d\n",cee_ver_id);
        switch(plat_id) {  
        case 3: printf("z/OS\n");
            break;
        case 4: printf("AS/400\n");
            break;
        default: printf("unrecognized platform id\n");
        }
    }
    return 0;
}
```

Figure 107. C/C++ Example of CEEGPID (Part 1 of 2)

```c
/* else use the old interface */
else {
    printf("The LE version is %d\n",cee_ver_id);
}
printf("The current platform is ");
switch(PLATFORM) {
    case 3: printf("z/OS\n");
        break;
    case 4: printf("AS/400\n");
        break;
    default: printf("unrecognized platform id\n");
}
```
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 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.

Figure 108. COBOL Example of CEEGPID (Part 1 of 2)
PROCEDURE DIVISION.
PARA-CBLGID.
** 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 108. COBOL Example of CEEGPID (Part 2 of 2)
CEEGQDT—Retrieve q_data_token

CEEGQDT retrieves the q_data_token from the Instance-Specific Information (ISI).

CEEGQDT is particularly useful when you have user-written condition handlers registered by CEEHDLR.
cond_rep (input)
A condition token defining the condition for which the q_data_token is retrieved.

q_data_token (output)
A pointer to the q_data associated with condition token cond_rep.

fc (output)
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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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 z/OS Language Environment Programming Guide.
• For more information about the CEEHDLR callable service, see “CEEHDLR—Register User-written condition handler” on page 325.
• For more information about the CEESGL callable service, see “CEESGL—Signal a condition” on page 449.
Examples

/*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 {
  int error_value;
  char err_msg_80;
  int retcode;
} info_struct;

int main(void) {
  _FEEDBACK fc,condtok;
  _ENTRY routine;
  _INT4 token,qdata;
  _INT2 c_1,c_2,cond_case,sev,control;
  _CHAR3 facid;
  _INT4 isi;
  info_struct *info;

  /* register the condition handler */
  token = 99;
  routine.address = (_POINTER)&handler;;
  routine.nesting = NULL;
  CEEHDLR(&routine,&token,&fc);
  if (_FBCHECK (fc, CEE000 ) != 0 ) {
    printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
    exit(2999);
  } /* . */

  /* set up the condition info structure */
  info = (info_struct *)malloc(sizeof(info_struct));
  if (info == NULL) {
    printf("error allocating info_struct\n");
    exit(2399);
  } /* . */

Figure 110. C/C++ Example of CEEGQDT (Part 1 of 2)
CEEGQDT

info->error_value = 86;
strcpy(info->err_msg,"Test message");
info->retcode = 99;
/* set qdata to be the condition info structure */
qdata = (int*)info;
/* build the condition token */
c_1 = 3;
c_2 = 99;
cond_case = 1;
sev = 3;
control = 0;
memcpy(facid,"ZZZ",3);
isi = 0;
CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,;
    facid,&isi,&condtok,&fc);
if (_FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEENCOD failed with message number %d\n",
        fc.tok_msgno);
    exit(2999);
}
/* signal the condition */
CEESGL(&condtok,&qdata,&fc);
if (_FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEESGL failed with message number %d\n",
        fc.tok_msgno);
    exit(2999);
}

/* */
void handler(_FEEDBACK *fc,_INT4 *token,_INT4 *result,
    _FEEDBACK *newfc) {
    _FEEDBACK qdatafc;
    _INT4 idata;
    info_struct *qdata;
    /* */
    /* */
    /* get the q_data_token from the ISI */
    CEEGQDT(fc, &idata, &qdatafc);
    if (_FBCHECK ( qdatafc , CEE000 ) != 0 ) {
        printf("CEEGQDT failed with message number %d\n",
            qdatafc.tok_msgno);
        *result = 20; /* percolate */
        return;
    }
    /************************************************************************/
    /* set info_struct pointer to address return by */
    /* CEEGQDT */
   ************************************************************************/
    qdata = (info_struct *) idata;
    /* use the condition info structure (qdata) */
    if (qdata->error_value == 86) {
        printf("%.12s\n",qdata->err_msg);
        printf("retcode = %d\n",qdata->retcode);
        *result = 10; /* resume this is what we want */
        return;
    }
    /************************************************************************/
    /************************************************************************/
    *result = 20; /* percolate */
}

Figure 110. C/C++ Example of CEEGQDT (Part 2 of 2)
### Figure 111. COBOL Example of CEEGQDT (Part 1 of 3)

```cobol
CBL LIB,QUOTE
*Module/File Name: IGZTGQDT
********************************************
** DRVGQDT - Drive sample program for CEEGQDT **
********************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRVGQDT.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE PROCEDURE-POINTER.
  01 TOKEN PIC S9(9) BINARY.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
          03 Case-Sev-Ctl PIC X.
          03 Facility-ID PIC XXX.
          02 I-S-Info PIC S9(9) BINARY.
        01 QDATA PIC S9(9) BINARY.
      01 CONDTOK.
      02 Condition-Token-Value.
        COPY CEEIGZCT.
          03 Case-1-Condition-ID.
            04 Severity PIC S9(4) BINARY.
            04 Msg-No PIC S9(4) BINARY.
          03 Case-2-Condition-ID
            REDEFINES Case-1-Condition-ID.
            04 Class-Code PIC S9(4) BINARY.
            04 Cause-Code PIC S9(4) BINARY.
            03 Case-Sev-Ctl PIC X.
            03 Facility-ID PIC XXX.
            02 I-S-Info PIC S9(9) BINARY.
```

---

**Chapter 4. Language Environment callable services** 313
** Register handler
SET ROUTINE TO ENTRY "CBLGQDT".
CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEEHDLR failed with msg "
      Msg-No of FC UPON CONSOLE
   STOP RUN
END-IF.
** Signal a condition
MOVE 1 TO QDATA.
SET CEE001 of CONDTOK to TRUE.
MOVE ZERO to I-S-Info of CONDTOK.
CALL "CEESGL" USING CONDTOK, QDATA, FC.
IF CEE000 of FC THEN
   DISPLAY "**** Resumed execution in the 
      "routine which registered the handler"
ELSE
   DISPLAY "CEESGL failed with msg "
      Msg-No of FC UPON CONSOLE
END-IF.
** UNregister handler
CALL "CEEHDLU" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEEHDLU failed with msg "
      Msg-No of FC UPON CONSOLE
END-IF.
STOP RUN.
END PROGRAM DRVGQDT.

******************************************************************************
** CBLGQDT - Call CEEQDT to get **
** the Q_DATA_TOKEN **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLGQDT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
         03 Case-1-Condition-ID.
            04 Severity PIC S9(4) BINARY.
            04 Msg-No PIC S9(4) BINARY.
         03 Case-2-Condition-ID
            REDEFINES Case-1-Condition-ID.
            04 Class-Code PIC S9(4) BINARY.
            04 Cause-Code PIC S9(4) BINARY.
         03 Case-Sev-Ctl PIC X.
         03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.
   01 QDATA PIC S9(9) BINARY.

Figure 111. COBOL Example of CEEQDT (Part 2 of 3)
The following example uses a COBOL program and handler to establish the condition handling environment prior to calling a PL/I subroutine to illustrate the use of the callable service from PL/I.

**Figure 111. COBOL Example of CEEQGQT (Part 3 of 3)**

The following example uses a COBOL program and handler to establish the condition handling environment prior to calling a PL/I subroutine to illustrate the use of the callable service from PL/I.
CBL LIB,QUOTE

*Module/File Name: IGZTGQDP

*************************************************
** IGZTGQDP - Drive sample program for CEEGQDT **
*************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGQDP.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE PROCEDURE-POINTER.
  01 TOKEN PIC S9(9) BINARY.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.
  01 FC.
  02 Condition-Token-Value.
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
  04 Severity PIC S9(4) BINARY.
  04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
   REDEFINES Case-1-Condition-ID.
  04 Class-Code PIC S9(4) BINARY.
  04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.
  01 QDATA PIC S9(9) BINARY.
  01 CONDTOK.
  02 Condition-Token-Value.
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
  04 Severity PIC S9(4) BINARY.
  04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
   REDEFINES Case-1-Condition-ID.
  04 Class-Code PIC S9(4) BINARY.
  04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
** Register handler

Figure 112. PL/I Example of CEEGQDT (Part 1 of 3)
SET ROUTINE TO ENTRY "HDLGQDT".
CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEEHDLR failed with msg "
   Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.

** Signal a condition
MOVE 1 TO QDATA.
SET CEE001 of CONDTOK to TRUE.
MOVE ZERO to I-S-Info of CONDTOK.
CALL "CEESGL" USING CONDTOK, QDATA, FC.
IF CEE000 of FC THEN
   DISPLAY "**** Resumed execution in the "
   "routine which registered the handler"
ELSE
   DISPLAY "CEESGL failed with msg "
   Msg-No of FC UPON CONSOLE
END-IF.

** UNregister handler
CALL "CEEHDLU" USING ROUTINE, FC.
IF NOT CEE000 of FC THEN
   DISPLAY "CEEHDLU failed with msg "
   Msg-No of FC UPON CONSOLE
END-IF.
STOP RUN.
END PROGRAM IGZTGQDP.

***********************************************************************
** HDLQDT -- COBOL condition handler to call **
** PL/I routine for actual work.  **
***********************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. HDLQDT.
DATA DIVISION.
LINKAGE SECTION.
01 CURCOND.
   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 TOKEN PIC 9(9) BINARY.
01 RESULT PIC 9(9) BINARY.
01 NEWCOND.
   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

Figure 112. PL/I Example of CEEGQDT (Part 2 of 3)
**Invoke the PL/I routine to handle condition**

CALL "IBMGQDT"
    USING ADDRESS OF CURCOND,
    ADDRESS OF TOKEN,
    ADDRESS OF RESULT,
    ADDRESS OF NEWCOND.
GOBACK.

END PROGRAM HDLGQDT.

Figure 112. PL/I Example of CEEGQDT (Part 3 of 3)
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 113. COBOL Example of CEEGQDT (Part 1 of 2)
CEEGTST

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 CEEESGL 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>
</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 HEAP(,,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 33 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 449.

- For more information about the CEEFRST callable service, see "CEEFRST—Free heap storage" on page 283.

- For more information about the CEEDSHP callable service, see "CEEDSHP—Discard heap" on page 256.
Examples

/*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 114. C/C++ Example of CEEGTST
CBL LIB,QUOTE
*Module/File Name: IGZTGTST
***************************************************
** IGZTGTST - Call CEEGTST to get heap storage  **
**  **
** In this example, a call is made to CEEGTST to **
** obtain 4000 bytes of storage from the initial **
** heap (HEAPID=0).  **
**
***************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGTST.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 HEAPID      PIC S9(9) BINARY.
  01 STGSIZE    PIC S9(9)  BINARY.
  01 ADDRSS     POINTER.
  01 FC.
     02 Condition-Token-Value.
        COPY CEEIGZCT.
         03 Case-1-Condition-ID.
             04 Severity   PIC S9(4) BINARY.
             04 Msg-No     PIC S9(4) BINARY.
         03 Case-2-Condition-ID
             REDEFINES Case-1-Condition-ID.
             04 Class-Code  PIC S9(4) BINARY.
             04 Cause-Code  PIC S9(4) BINARY.
         03 Case-Sev-Ctl
             03 Facility-ID PIC XXX.
     02 I-S-Info
        PIC S9(9)  BINARY.
PROCEDURE DIVISION.
***************************************************
** Specify 0 to get storage from initial heap. **
** Specify 4000 to get 4000 bytes of storage. **
** Call CEEGTST to obtain storage.  **
***************************************************
PARA-CBLGTST.
  MOVE 0 TO HEAPID.
  MOVE 4000 TO STGSIZE.
  CALL "CEEGTST" USING HEAPID, STGSIZE,
          ADDRSS, FC.
  IF CEE000 of FC THEN
     DISPLAY "Obtained " STGSIZE " bytes of"
     " storage at location " ADDRSS
     " from heap number " HEAPID
  ELSE
     DISPLAY "CEEGTST failed with msg "
     Msg-No of FC UPON CONSOLE
     STOP RUN
  END-IF.
GOBACK.

Figure 115. COBOL Example of CEEGTST
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.

```plaintext
*PROCESS MACRO;
/* Module/FileName: 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 CEEIBMAN;
%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;
```

Figure 116. PL/I Example

**CEEHDLR—Register User-written condition handler**

CEEHDLR registers a user-written condition handler for the current stack frame. The user condition handler is invoked when:

- It is registered for the current stack frame by CEEHDLR, and
- The Language Environment condition manager requests the condition handler associated with the current stack frame handle the condition.

Language Environment places the user-written condition handlers associated with each stack frame in a queue. The queue can be empty at any given time. The Language Environment condition manager invokes the registered condition handlers in LIFO (last in, first out) order to handle the condition.
The opposite of CEEHDLR, which registers a user-written condition handler, is CEEHDLU, which unregisters the handler. You do not necessarily need to use CEEHDLU to remove user-written condition handlers you registered with CEEHDLR. Any user-written condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by Language Environment, but only when the associated stack frame is removed from the stack.

Language Environment condition handlers are driven only for synchronous conditions.

### Syntax

```plaintext
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 335.

Examples

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

#ifdef __cplusplus
extern "C"
{
#endif

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

#ifdef __cplusplus
}
#endif

int main(void)
{
    _FEEDBACK fc;
    _ENTRY routine;
    _INT4 token;

    /* set the routine structure to point to the handler */
    /* and use CEEHDLR to register the user handler */

    token = 99;
    routine.address = (_POINTER)&handler;
    routine.nesting = NULL;

    CEEHDLR(&routine,&token,&fc);

    /* verify that CEEHDLR was successful */
    if (_FBCHECK ( fc, CEE000 ) != 0 ) {
        printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
        exit (2999);
    }
    /*
    :
    */

    /*************************************************************************/
    /* handler is a user condition handler */
    /*************************************************************************/
    void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
                 _FEEDBACK *newfc) {
        /*
        :
        */
```
CBL LIB,QUOTE
*Module/File Name: IGZTHDLR
*******************************************************************************
** CBLHDLR - Call CEEHDLR to register a user **
** condition handler **
*******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLHDLR.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ROUTINE PROCEDURE-POINTER.
  01 TOKEN PIC S9(9) BINARY.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.
  01 QDATA PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
  01 CONDTOK.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLHDLR.
  SET ROUTINE TO ENTRY "HANDLER".
  CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
  IF NOT CEE0000 of FC THEN
    DISPLAY "CEEHDLR failed with msg "
      Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.

Figure 118. COBOL Program that Registers HANDLER Routine (Part 1 of 2)
* RAISE A SIGNAL

PARA-CBLGL.

*******************************************************************************
** 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 **
*******************************************************************************

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 118. COBOL Program that Registers HANDLER Routine (Part 2 of 2)
CBL LIB,QUOTE,NOOPT,NODYNAM
*Module/File Name: IGZTHAND
*************************************************
** DRVHAND - Drive sample program for COBOL **
** user-written condition handler. **
**
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRVHAND.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE                  PROCEDURE-POINTER.
  01 DENOMINATOR  PIC S9(9) BINARY.
  01 NUMERATOR  PIC S9(9) BINARY.
  01 RATIO     PIC S9(9) BINARY.
  01 TOKEN     PIC S9(9) BINARY VALUE 0.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No    PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
    02 I-S-Info    PIC S9(9) BINARY.
PROCEDURE DIVISION.
REGISTER-HANDLER.
*************************************************
** Register handler **
*************************************************
SET ROUTINE TO ENTRY "HANDLER".
CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
IF NOT CEE000 of FC THEN
  DISPLAY "CEEHDLR failed with msg "
    Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.
RAISE-CONDITION.
*************************************************
** Cause a zero-divide condition. **
*************************************************
MOVE 0 TO DENOMINATOR.
MOVE 1 TO NUMERATOR.
DIVIDE NUMERATOR BY DENOMINATOR
GIVING RATIO.

Figure 119. COBOL User-Written Condition Handler Registered by CBLHDLR and Unregistered by CBLHDLU (Part 1 of 2)
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.
      08 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.
  GORACK.
END PROGRAM HANDLER.

Figure 119. COBOL User-Written Condition Handler Registered by CBLHDLR and
Unregistered by CBLHDLU (Part 2 of 2)
*Process macro;
/* Module/File Name: IBMHDLR */
/***************************************************************************/
/* */
/* EXCOND .-> DIVZERO */
/* - register handler | force a divide-by-0 */
/* - call DIVZERO --' */
/* => "resume point" */
/* - unregister handler */
/* */
/* USRHDLR: */
/* - if divide-by-zero then */
/* - move resume cursor */
/* - resume at "resume" */
/* */
/* */
/***************************************************************************/
Excond :Proc Options(Main);
*******************************************************************************/
/* Important elements are found in these includes */
/* - feedback declaration */
/* - fbcheck macro call */
/* - condition tokens such as CEE000 */
/* - entry declarations such as ceehdlr */
/*******************************************************************************/
%include ceeibmct;
%include ceeibmaw;

dcl Usrhdlr external entry;

dcl 1 fback feedback;
dcl divisor fixed bin(31);
dcl token fixed bin(31);

/*******************************************************************************/
/* Register a user-written condition handler */
/*******************************************************************************/
token = 97;
Call ceehdlr(Usrhdlr, token, fback);
If fbcheck (fback, cee000) then
  display ('MAIN: registered USRHDLR');
else
  do;
    display ('CEEHDLR failed with message number ' ||
      fback.MsgNo);
    stop;
  end;

Figure 120. EXCOND Program (PL/I) to Handle Divide-by-Zero Condition (Part 1 of 2)
/*********************************************/  
/* Call DIVZERO to divide by zero */  
/* and drive USRHDLR */  
/*********************************************/  
divisor = 0;  
call divzero (divisor);  
display ('MAIN: resumption after DIVZERO');  
  
/*********************************************/  
/* Unregister the user condition handler */  
/*********************************************/  
Call ceehdlu (Usrhdlr, fback);  
If fbcheck (fback, cee000) then  
display ('MAIN: unregistered USRHDLR');  
else  
do;  
  display ('CEEHDLU failed with message number ' ||  
     fback.MsgNo);  
  stop;  
end;  
  
/*********************************************/  
/* Subroutine that simply raises ZERODIVIDE */  
/*********************************************/  
divzero: proc (arg);  
dcl arg fixed bin(31);  
  display(' DIVZERO: starting.');  
  arg = 1 / arg;  
  display(' DIVZERO: Returning to its caller');  
  end divzero;  
end divzero;  
end Excond;  

Figure 120. 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=DSASIZ, Main memory to obtain  *
MAIN=YES, This program is a MAIN prog *
PPA=PPA1, Our Program Prolog Area *
BASE=R10 Base reg for executable code
USING CEECAA,R12 Addressing for LE/370 CAA
USING CEEDSA,R13 Addressing for dynamic data
*
* Announce ourselves
*
WTO 'CEEHDLRA Says "HELLO",ROUTCDE=11
*
* Register User Handler
*
LA  R1,USRHDLPP Get addr of proc-ptr to Hdlr
ST  R1,PARM1 Make it 1st parameter
LA  R1,TOKEN Get addr of 32-bit token
ST  R1,PARM2 Make it 2nd parameter
LA  R1,FEEDBACK Get addr of Feedback Code
ST  R1,PARM3 Make it 3rd parameter
LA  R1,HDLRPLST Point to CEEHDLR's parm list
CALL CEEHDLR Invoke CEEHDLR service
CLC FEEDBACK,=XL12'00' Check for success..
BE HDLRGOOD Skip diagnostics if success
*
WTO '**** Call to CEEHDLR Failed ****',ROUTCDE=11
*
ABEND 1,DUMP Terminate program with Dump

Figure 121. PL/I Example of CEEHDLR (Part 1 of 2)
CEEHDLU—Unregister user-written condition handler

CEEHDLU unregisters a user condition handler for the current stack frame.

You do not necessarily need to use CEEHDLU to remove user-written condition handlers you registered with CEEHDLR. Any user-written condition handlers created through CEEHDLR and not unregistered by CEEHDLU are unregistered automatically by Language Environment, but only when the associated stack frame is removed from the stack.
Note: For information about restrictions on the use of CEEHDLU with PL/I, see z/OS Language Environment Programming Guide.

Syntax

```
CEEHDLU(---routine---,---fc---)
```

**routine (input)**

An entry variable or constant for the routine to be unregistered as a user condition handler. This routine must be previously registered (with CEEHDLR) by the same stack frame that invokes CEEHDLU.

**fc (output)**

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

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 325 for further information about specifying the **routine** parameter.
/*Module/File Name: EDCHDLU  */

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

#ifdef __cplusplus
    extern "C" {
#endif

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

#ifdef __cplusplus
}
#endif

int main(void)
{
    _FEEDBACK fc;
    _ENTRY routine;
    _INT4 token;

    /* set the routine structure to point to the handler */
    /* and use CEEHDLR to register the user handler */

    token = 99;
    routine.address = (_POINTER)&handler;;
    routine.nesting = NULL;

    CEEHDLR(&routine, &token, &fc);

    /* verify that CEEHDLR was successful */
    if ( _FBCHECK ( fc, CEE000 ) != 0 ) {
        printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
        exit (2999);
    }
    /* */
    /* */
    /* Unregister the condition handler */
    CEEHDLU(&routine, &fc);

    /* verify that CEEHDLU was successful */
    if ( _FBCHECK ( fc, CEE000 ) != 0 ) {
        printf("CEEHDLU failed with message number %d\n", fc.tok_msgno);
        exit (2999);
    }
    /* */
    /* */
    /* */
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result,
            _FEEDBACK *newfc) {
    /* */
    /* */
}

Figure 122. C/C++ Example of CEEHDLU
** CBLHDLU - Call CEEHDLU to unregister a user condition handler **
** In this example, a call is made to CEEHDLU **
** to unregister a user condition handler **
** previously registered using CEEHDLR. **
**
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLHDLU.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ROUTINE     PROCEDURE-POINTER.
01 TOKEN       PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
   COPY CEEIGZCT.
      03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No    PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
   02 I-S-Info    PIC S9(9) BINARY.

PROCEDURE DIVISION.
PARA-CBLHDLR.
   SET ROUTINE TO ENTRY "HANDLER".
   CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
   IF NOT CEE000 of FC THEN
      DISPLAY "CEEHDLR failed with msg "
         Msg-No of FC UPON CONSOLE
      STOP RUN
   ELSE
      DISPLAY "HANDLER REGISTERED"
   END-IF.

* ...

Figure 123. COBOL Program that Unregisters User-Written Condition Handler (Part 1 of 2)
PARA-CBLHDLU.
   CALL "CEEHDLU" USING ROUTINE, FC.
   IF NOT CEE000 of FC THEN
      DISPLAY "CEEHDLU failed with msg "
      Msg-No of FC UPON CONSOLE
      STOP RUN
   ELSE
      DISPLAY "HANDLER UNREGISTERED"
   END-IF.
   GOBACK.
END PROGRAM CBLHDLU.

Figure 123. COBOL Program that Unregisters User-Written Condition Handler (Part 2 of 2)
**Module/File Name: IGGTHAND**

*************************************************

** 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.

Figure 124. COBOL User-Written Condition Handler Registered by CBLHDLR and Unregistered by CBLHDLU (Part 1 of 2)
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 "
  MsgBox-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.
     08 RESUME VALUE 10.
01 CURCOND.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
      04 Severity PIC S9(4) BINARY.
      04 MsgBox-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-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 MsgBox-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) BINARY.
      04 Cause-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 " MsgBox-No Of CURCOND
  " -- will resume execution".
  SET RESUME TO TRUE.

GORTACK.
END PROGRAM HANDLER.

Figure 124. 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**

```plaintext
CEEISEC(input_year, input_months, input_day, input_hours, input_minutes, input_seconds, input_milliseconds)
```

**input_year (input)**
A 32-bit binary integer representing the year.

The range of valid `input_year` is 1582 to 9999, inclusive.

**input_month (input)**
A 32-bit binary integer representing the month.

The range of valid `input_month` is 1 to 12.

**input_day (input)**
A 32-bit binary integer representing the day.

The range of valid `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>
<tr>
<td>CEE2EH</td>
<td>3</td>
<td>2513</td>
<td>The input date passed in a CEEISEC, CEEEDAYS, 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 435.
- See "CEESECI—Convert seconds to integers" on page 430 for more information about the CEESECI callable service.
Examples

/*Module/File Name: EDCISEC */

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

int main(void) {

_INT4 year, month, day, hours, minutes, seconds,
    millisecs;
_FLOAT8 output;
_FEEDBACK fc;

year = 1991;
month = 9;
day = 13;
hours = 4;
minutes = 34;
seconds = 25;
millisecs = 746;

CEEISEC(&year,&month,&day,&hours,&minutes,&seconds,;
        &millisecs,&output,&fc);

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

printf("The number of seconds between 00:00:00.00"
       " 10/14/1582 and 04:34:25.746 09/13/1991"
       " is %.3f\n",output);
}

Figure 125. C/C++ Example of CEEISEC
CBL LIB,QUOTE
*Module/File Name: IGZTISEC
*******************************************************************************
**
** CBLISEC - Call CEEISEC to convert integers **
** to seconds **
**
*******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLISEC.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 YEAR PIC S9(9) BINARY.
  01 MONTH PIC S9(9) BINARY.
  01 DAYS PIC S9(9) BINARY.
  01 HOURS PIC S9(9) BINARY.
  01 MINUTES PIC S9(9) BINARY.
  01 SECONDS PIC S9(9) BINARY.
  01 MILLSEC PIC S9(9) BINARY.
  01 OUTSECS COMP-2.
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLISEC.
*******************************************************************************
** Specify seven binary integers representing **
** the date and time as input to be converted **
** to Lillian 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.

Figure 126. COBOL Example of CEEISEC (Part 1 of 2)
CALL "CEEISEC" USING YEAR, MONTH, DAYS, 
HOURS, MINUTES, SECONDS, 
MILLSEC, OUTSECS, FC. 

** If CEEISEC runs successfully, display result**
******************************************************************************
IF CEE000 of FC THEN 
  DISPLAY MONTH "/" DAYS "/" YEAR 
  " AT " HOURS ":" MINUTES ":" SECONDS 
  " is equivalent to " OUTSECS " seconds"
ELSE 
  DISPLAY "CEEISEC failed with msg 
  Msg-No of FC UPON CONSOLE 
  STOP RUN ENDF. 
END-IF. 

GOBACK.
**PROCESS MACRO;**
/*Module/File Name: IBMISEC**
*****************************************************/
/** Function: CEEISEC - Convert integers to **
/** seconds **
/** **
/** In this example, CEEISEC is called to convert **
/** integers representing the date and time to the **
/** number of seconds since 00:00 14 October 1582. **
/** **
*****************************************************/
PLIISEC: PROC OPTIONS(MAIN);
%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL YEAR REAL FIXED BINARY(31,0);
DCL MONTH REAL FIXED BINARY(31,0);
DCL DAYS REAL FIXED BINARY(31,0);
DCL HOURS REAL FIXED BINARY(31,0);
DCL MINUTES REAL FIXED BINARY(31,0);
DCL SECONDS REAL FIXED BINARY(31,0);
DCL MILLSEC REAL FIXED BINARY(31,0);
DCL OUTSECS REAL FLOAT DECIMAL(16);
DCL 01 FC, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

/* Specify integers representing */
/* 00:00:00 1 January 2000 */

YEAR = 2000;
MONTH = 1;
DAYS = 1;
HOURS = 0;
MINUTES = 0;
SECONDS = 0;
MILLSEC = 0;

Figure 127. PL/I Example of CEEISEC (Part 1 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**

```
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 400, for more information about the CEENCOD callable service.
Examples

/*Module/File Name: EDCITOK */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <leawi.h>
#include <ceeedct.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 128. C/C++ Example of CEEITOK (Part 1 of 2)
CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,
    facid,&isi,&condtok,&fc);

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

/*
 * signal the condition */
CEESGL(&condtok,&qdata,&fc);
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEESGL failed with message number %d\n",
            fc.tok_msgno);
    exit(2999);
}

*/

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result, 
    _FEEDBACK *newfc) {

    _FEEDBACK orig_fc, itok_fc;

    /*
    */
    /* get the original condition token */
    CEEITOK(&orig_fc, &itok_fc);
    if ( _FBCHECK ( itok_fc , CEE000 ) != 0 ) {
        printf("CEEITOK failed with message number %d\n",
                itok_fc.tok_msgno);
        exit(2999);
    }
    /*
    */
    *result = 10;
}

Figure 128. C/C++ Example of CEEITOK (Part 2 of 2)
CBL LIB,QUOTE,NOOPT
*Module/File Name: IGZTITOK
******************************************************************************
** Purpose: Drive sample program for CEEITOK. **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRVITOK.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE PROCEDURE-POINTER.
  01 DENOMINATOR PIC S9(9) BINARY.
  01 NUMERATOR PIC S9(9) BINARY.
  01 RATIO PIC S9(9) BINARY.
  01 TOKEN PIC S9(9) BINARY VALUE 0.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
          03 Case-2-Condition-ID
            REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
        02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
REGISTER-HANDLER.
******************************************************************************
** Register handler **
******************************************************************************
  SET ROUTINE TO ENTRY "CBLITOK".
  CALL "CEEHDLR" USING ROUTINE, TOKEN, FC.
  IF NOT CEE000 of FC THEN
    DISPLAY "CEEHDLR failed with msg "
      Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
RAISE-CONDITION.
******************************************************************************
** Cause a zero-divide condition. **
******************************************************************************
  MOVE 0 TO DENOMINATOR.
  MOVE 1 TO NUMERATOR.
  DIVIDE NUMERATOR BY DENOMINATOR, GIVING RATIO.

Figure 129. 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. CEEITOK.
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-Seq-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-Seq-Ctl PIC X.
           03 Facility-ID PIC XXX.
       02 I-S-Info PIC S9(9) BINARY.

Figure 129. COBOL Example of CEEITOK (Part 2 of 3)
LINKAGE SECTION.
01 TOKEN PIC S9(9) BINARY.
01 RESULT PIC S9(9) BINARY.
B8 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 FC
   ELSE
      DISPLAY "CEEITOK failed with msg "
      Msg-No of FC UPON CONSOLE
      STOP RUN
   END-IF.
PARA-HANDLER.
*****************************************************************************
** In user handler - resume execution  **
*****************************************************************************
SET RESUME TO TRUE.

GOBACK.

END PROGRAM CBLITOK.

Figure 129. COBOL Example of CEEITOK (Part 3 of 3)
CEELCNV—Query Locale numeric conventions

CEELCNV, analogous to the C function `localeconv()`, queries the numeric formatting information from the current locale and sets the components of a structure with values pertaining to the LC_NUMERIC and LC_MONETARY categories. It sets the components of an object of the type NM_STRUCT with the values appropriate for the formatting of the numeric quantities (monetary and otherwise) according to the rules of the current locale.

CEELCNV is sensitive to the locales set by `setlocale()` or CEESETL, not to the Language Environment settings from COUNTRY or CEE3CTY.
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 105.

num_and_mon (output)
Points to the numeric and monetary structure that is filled in by this service. If the service fails, the contents of the structure are undefined.

num_and_mon has the following structure:

NM_STRUCT
A halfword length-prefixed character string (VSTRING). A pointer to the filled-in structure NM_STRUCT is returned. The structure pointed to by the return value should not be modified by the program but can be overridden by subsequent calls to CEELCNV. In addition, calls to CEESETL with the LC_ALL, LC_MONETARY or LC_NUMERIC categories can cause subsequent calls to CEELCNV to return different values based on the selection of the locale.

The members of the structure with the type VSTRING are strings, any of which (except decimal_point) can point to an empty string, to indicate that the value is not available in the current locale or is of zero length. The members with type VINT are non-negative numbers, any of which can be CHAR_MAX to indicate that the value is not available in the current locale. The members include the following:

VSTRING decimal_point
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the decimal-point character used to format non-monetary quantities.

VSTRING thousands_sep
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the character used to separate groups of digits to the left of the decimal point in formatted non-monetary quantities.

VSTRING grouping
A halfword length-prefixed character string (VSTRING) of 22 bytes whose elements indicate the size of each group of digits in formatted non-monetary quantities.

VSTRING int_curr_symbol
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the international currency symbol applicable to the current locale, left justified within a four-character space-padded field.

VSTRING currency_symbol
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the local currency symbol applicable to the current locale.

VSTRING mon_thousands_sep
A halfword length-prefixed character string (VSTRING) of 22 bytes that is the separator for groups of digits to the left of the decimal point in formatted monetary quantities.
VSTRING mon_grouping
A halfword length-prefixed character string (VSTRING) of 22 bytes whose elements indicate the size of each group of digits in formatted monetary quantities.

VSTRING positive_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes that indicates a non-negative-formatted monetary quantity.

VSTRING negative_sign
A halfword length-prefixed character string (VSTRING) of 22 bytes used to indicate a negative-formatted monetary quantity.

VINT int_frac_digits
A 1-byte integer that is the number of fractional digits (those to the right of the decimal point) to be displayed in an internationally-formatted monetary quantity.

VINT frac_digits
A 1-byte integer that is the number of fractional digits (those to the right of the decimal point) to be displayed in a formatted monetary quantity.

VINT p_cs_precedes
A 1-byte integer that is set to 1 if the currency_symbol precedes the value for a non-negative-formatted monetary quantity. It is set to 0 if it follows.

VINT p_sep_by_space
A 1-byte integer that is set to 1 if the currency_symbol is separated by a space from the value for a non-negative-formatted monetary quantity. It is set to 0 if it is not separated.

VINT n_cs_precedes
A 1-byte integer that is set to 1 if the currency_symbol precedes the value for a negative-formatted monetary quantity. It is set to 0 if it follows.

VINT n_sep_by_space
A 1-byte integer that is set to 1 if the currency_symbol is separated by a space from the value for a negative-formatted monetary quantity. It is set to 0 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.

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—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 416 for a description of LC_NUMERIC and LC_MONETARY categories.
- See “CEESETL—Set locale operating environment” on page 443 for details on the CEESETL callable service.
Examples

CBL LIB,QUOTE
*Module/FILE Name: IGZTLCNV
******************************************************************************
** Example for callable service CEELCNV  **
** Function: Retrieve numeric and monetary  **
** format for default locale and  **
** print an item.  **
** Set locale to France, retrieve  **
** structure, and print an item.  **
** Valid only for COBOL for MVS & VM Release 2  **
** or later.  **
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINLCNV.
DATA DIVISION.
WORKING-STORAGE SECTION.
******************************************************************************
** Use Locale NM-Struct for CEELCNV calls  **
******************************************************************************
COPY CEEIGZNM.
*
PROCEDURE DIVISION.
******************************************************************************
** Subroutine needed for addressing  **
******************************************************************************
CALL "COBLCNV" USING NM-Struct.
STOP RUN.
*
IDENTIFICATION DIVISION.
PROGRAM-ID. COBLCNV.
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 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 131. COBOL Example of CEELCNV (Part 1 of 2)
LINKAGE SECTION.
***********************************************************************
** Use Locale NM-Struct for CEELCNV calls **
*************************************************************************
COPY CEEIGZNM.
*
PROCEDURE DIVISION USING NM-Struct.
*************************************************************************
** Call CEELCNV to retrieve values for locale**
*************************************************************************
CALL "CEELCNV" USING OMITTED,
       ADDRESS OF NM-Struct, FC.
*************************************************************************
** Check feedback code and display result **
*************************************************************************
IF Severity = 0 THEN
   DISPLAY "Default decimal point is "
       DECIMAL-PT-String(1:DECIMAL-PT-Length)
ELSE
   DISPLAY "Call to CEELCNV failed. " Msg-No
END-IF.
*************************************************************************
** Set up locale for France **
*************************************************************************
MOVE 4 TO LN-Length.
MOVE "FRAN" TO LN-String (1:LN-Length).
*************************************************************************
** CallCEESETL to set monetary locale **
*************************************************************************
CALL "CEESETL" USING Locale-Name,
       LC-MONETARY, FC.
*************************************************************************
** Call CEESETL to set numeric locale **
*************************************************************************
CALL "CEESETL" USING Locale-Name,
       LC-NUMERIC, FC.
*************************************************************************
** Check feedback code and call CEELCNV again **
*************************************************************************
IF Severity = 0
   CALL "CEELCNV" USING OMITTED,
          ADDRESS OF NM-Struct, FC
   IF Severity > 0
      DISPLAY "Call to CEELCNV failed. "
          Msg-No
   ELSE
      DISPLAY "French decimal point is "
          DECIMAL-PT-String(1:DECIMAL-PT-Length)
   END-IF
ELSE
   DISPLAY "Call to CEESETL failed. " Msg-No
END-IF.
EXIT PROGRAM.
END PROGRAM COBLCNV.
*
END PROGRAM MAINLCNV.

Figure 131. COBOL Example of CEELCNV (Part 2 of 2)
CEEOCT—Get current local date or time

CEEOCT 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.

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

```plaintext
CEEOCT(--output_Lilian--,--output_seconds--,--output_Gregorian--,---)
```

**output_Lilian (output)**

A 32-bit binary integer representing the current local date in the Lilian format, that is, day 1 equals 15 October 1582, day 148,887 equals 4 June 1990. If the local time is not available from the system, output_Lilian is set to 0 and CEELOCT terminates with a non-CEE000 symbolic feedback code.

**output_seconds (output)**

A 64-bit double-floating point number representing the current local date and time as the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). 19:00:01.078 on 4 June 1990 is second number 12,863,905,201.078.

If the local time is not available from the system, output_seconds is set to 0 and CEELOCT terminates with a non-CEE000 symbolic feedback code.

**output_Gregorian (output)**

A 17-byte fixed-length character string in the form YYYYMMDDHHMISS999 representing local year, month, day, hour, minute, second, and millisecond.

If the format of output_Gregorian does not meet your needs, you can use the CEEDATM callable service to convert output_seconds to another format.

**fc (output)**

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

The following 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 CEELOCT.
CEEOCT

- CICS consideration—CEEOCT does not use the OS TIME macro.
- z/OS UNIX consideration—In multithread applications, CEEOCT affects only the calling thread.

**For more information**
- See "CEEDATM—Convert seconds to character timestamp" on page 234 for more information about the CEEDATM callable service.

**Examples**

```c
/*Module/File Name: EDCLOCT */

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

int main(void) {
  _FEEDBACK fc;
  _INT4 lil_date;
  _FLOAT8 local_date;
  _CHAR17 gregorian_date;
  CEEOCT(&lil_date,&local_date,gregorian_date,&fc);
  if (_FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEEOCT failed with message number %d\n", fc.tok_msgno);
    exit(2999);
  }
  printf("The current date is YYYMMDDHHMISS999\n");
  printf("%.17s\n",gregorian_date);
}
```

*Figure 133. C/C++ Example of CEEOCT*
**Module/File Name: CBLLOCT**
**-----------------------------**
CBLLOCT - Call CEELOCT to get current **
** local time **
**
** In this example, a call is made to CEELOCT **
** to return the current local time in Lilian **
** days (the number of days since 14 October **
** 1582), Lilian seconds (the number of **
** seconds since 00:00:00 14 October 1582), **
** and a Gregorian string (in the form **
** YYYMMDDMISS999). The Gregorian character **
** string is then displayed. **
**
**
***********************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLLOCT.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 LILIAN PIC S9(9) BINARY.
  01 SECONDS COMP-2.
  01 GREGORN PIC X(17).
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl
          PIC X.
        03 Facility-ID PIC XXX.
        02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLLOCT.
  CALL "CEELOCT" USING LILIAN, SECONDS,
    GREGORN, FC.
***********************
** If CEELOCT runs successfully, display **
** Gregorian character string **
***********************
IF CEE000 of FC THEN
  DISPLAY "Local Time is " GREGORN
ELSE
  DISPLAY "CEELOCT failed with msg "
    Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.
GOBACK.

---

Figure 134. COBOL Example of CEELOCT
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

```plaintext
>>>CEEMGET(--cond_token--,--message_area--,--msg_ptr--,--fc--)<<<
```
cond_token (input)
A 12-byte condition token received as the result of a Language Environment callable service.

message_area (input/output)
A fixed-length 80-character string (VSTRING), where the message is placed.
The message is left-justified and padded on the right with blanks.

msg_ptr (input/output)
A 4-byte binary integer returned to the calling routine.
The msg_ptr should be passed a value of zero on the initial call to CEEMGET.
If a message is too large to be contained in the message_area, msg_ptr (containing the index) is returned into the message. This index is used on subsequent calls to CEEMGET to retrieve the remaining portion of the message. A feedback code is also returned, indicating the message has been truncated. When the entire message is returned, msg_ptr is zero.

msg_ptr contains different results based on the length of the message:
• If a message contains fewer than 80 characters, the entire message is returned on the first call. msg_ptr contains 0.
• If a message contains exactly 80 characters, the entire message is returned on the first call. msg_ptr contains 0.
• If the message is too long, CEEMGET splits it into segments. The msg_ptr 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 message-number and facility ID facility-id could not be located.</td>
</tr>
<tr>
<td>CEE0E6</td>
<td>3</td>
<td>0454</td>
<td>The message number message-number could not be found for facility ID facility-id.</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 400 for a description of the 12-byte condition token constructed by the CEENCOD callable service.

---

<table>
<thead>
<tr>
<th>Code</th>
<th>Severity</th>
<th>Message Number</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEE0E7</td>
<td>1</td>
<td>0455</td>
<td>The message with message number <em>message-number</em> and facility ID <em>facility-id</em> was truncated.</td>
</tr>
<tr>
<td>CEE0EA</td>
<td>3</td>
<td>0458</td>
<td>The message repository <em>repository-name</em> could not be located.</td>
</tr>
</tbody>
</table>
Examples

/*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);
        }
    }
}

Figure 136. C/C++ Example of CEEMGET (Part 1 of 2)
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);
}
}
}
}

Figure 136. 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 CEEIGZCT.
     03 Case-1-Condition-ID.
      04 Severity  PIC S9(4) BINARY.
      04 Msg-No    PIC S9(4) BINARY.
     03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-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.
  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 1-S-Info     PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLMGET.

Figure 137. COBOL Example of CEEMGET (Part 1 of 2)
**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.**
CEEMOUT—Dispatch a message

CEEMOUT dispatches a user-defined message string to the message file.
CEEMOUT

Syntax

```plaintext
CEEMOUT(message_string, destination_code, fc)
```

message_string (input)
A halfword-prefixed printable character string containing the message. DBCS characters must be enclosed within shift-out (byte X'0F') shift-in (X'0E') characters.

Insert data cannot be placed in the message with CEEMOUT. The halfword-prefixed message string (input) must contain only printable characters. For length greater than zero, unpredictable results will occur if the byte following the halfword prefix is X'00'.

destination_code (input)
A 4-byte binary integer. The only accepted value for destination_code is 2. Under systems other than CICS, Language Environment writes the message to the ddname of the file specified in the MSGFILE 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>CEE000</td>
<td>0</td>
<td>—</td>
<td>The service completed successfully.</td>
</tr>
<tr>
<td>CEE0E3</td>
<td>3</td>
<td>0451</td>
<td>An invalid destination code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>destination_code was passed to routine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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 47 for more information about the MSGFILE run-time option.
- See z/OS Language Environment Programming Guide for more information on CESE.
Examples

/*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 139. C/C++ Example of CEEMOUT
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLMOUT.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 MSGSTR.
    02 Vstring-length PIC S9(4) BINARY.
    02 Vstring-text.
      03 Vstring-char PIC X,
          OCCURS 0 TO 256 TIMES
          DEPENDING ON Vstring-length
          of MSGSTR.
  01 DESTIN PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
        03 Case-1-Condition-ID.
          04 Severity PIC S9(4) BINARY.
          04 Msg-No PIC S9(4) BINARY.
        03 Case-2-Condition-ID
          REDEFINES Case-1-Condition-ID.
          04 Class-Code PIC S9(4) BINARY.
          04 Cause-Code PIC S9(4) BINARY.
        03 Case-Sev-Ctl PIC X.
        03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLMOUT.
  *******************
  ** Create message string and specify length **
  *******************
  MOVE 25 TO Vstring-length of MSGSTR.
  MOVE "CEEMOUT ran successfully"
      TO Vstring-text of MSGSTR.
  *******************
  ** Specify 2 to send the message to the ddname **
  ** specified or defaulted in the MSGFILE **
  ** run-time option. **
  *******************
  MOVE 2 TO DESTIN.
  CALL "CEEMOUT" USING MSGSTR, DESTIN, FC.
  IF NOT CEE000 of FC THEN
    DISPLAY "CEEMOUT failed with msg "
      Msg-No of FC UPON CONSOLE
    STOP RUN
  END-IF.
  GOBACK.

Figure 140. COBOL Example of CEEMOUT
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}
```
### CEEMRCE

**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**

- See [z/OS Language Environment Programming Guide](#) for more information about the CEEMRCE callable service.
Examples

```cobol
CBL NODYNAM APOST
IDENTIFICATION DIVISION.
PROGRAM-ID. PGM2.
*Module/File Name: IGZTMRCE
*__________________________________________*
* Sample program using CEE3SRP and CEEMRCE.*
* PGM2 registers user-written condition *
* handler UCH1 using CEEHDLR. It *
* sets a resume point using CEE3SRP. After *
* incurring a condition and returning *
* to PGM2, PGM3 is called. PGM3 sets up *
* new resume point, does a divide-by-zero, *
* and after resuming in PGM3, resets the *
* resume point to PGM2 and does a GOBACK. *
*__________________________________________*

DATA DIVISION.
WORKING-STORAGE SECTION.
01 RECOVERY-AREA EXTERNAL.
   05 RECOVERY-POINT POINTER.
   05 ERROR-INDICATOR PIC X(01).

01 UCH-Routine PROCEDURE-POINTER.

01 FIELDS.
   05 FIRST-TIME-SW PIC X(03) VALUE ' ON'.
      88 FIRST-TIME-88 VALUE ' ON'.
   05 ANSWER PIC S9(02) COMP-3 VALUE 0.
   05 UCH1 PIC X(08) VALUE 'UCH1'.
   05 PGM3 PIC X(08) VALUE 'PGM3'.
   05 CEEMRCE PIC X(08) VALUE 'CEEMRCE'.
   05 CEE3SRP PIC X(08) VALUE 'CEE3SRP'.
   05 TOKEN PIC S9(09) BINARY.
   05 FC.
      10 CASE-1.
         15 SEVERITY PIC S9(04) BINARY.
         15 MSG-WO PIC S9(04) BINARY.
      10 SEV-CTL PIC X(01).
      10 FACILITY-ID PIC X(03).
      10 I-S-INFO PIC S9(09) BINARY.

PROCEDURE DIVISION.

SET UCH-Routine TO ENTRY 'UCH1'.
*__________________________________________*
* Register the condition handler, UCH1. *
*__________________________________________*

CALL CEEHDLR USING UCH-Routine, TOKEN, FC.
IF CASE-1 NOT = LOW-VALUE GOBACK.
PERFORM COMPUTE-LOOP 3 TIMES.
CALL PGM3 USING RECOVERY-AREA.
SET RECOVERY-POINT TO NULL.
GOBACK.

Figure 142. COBOL Example of CEEMRCE (Part 1 of 4)
```
COMPUTE-LOOP.
  IF FIRST-TIME-BB
    MOVE 'OFF' TO FIRST-TIME-SW
  * Set up a new resume point.
  ________________________________________________________________
    CALL CEE3SRP USING RECOVERY-POINT, CASE-1
    IF CASE-1 NOT = LOW-VALUE
      GOBACK.
    IF ERROR-INDICATOR = 'E'
      MOVE SPACE TO ERROR-INDICATOR
      MOVE 1 TO ANSWER.
  * Application code may go here.
    COMPUTE ANSWER = 1 / ANSWER.
  * Put application code here.
  ________________________________________________________________
CBL NODYNAM APOST
IDENTIFICATION DIVISION.
PROGRAM-ID. PGM3.
  * Sample program using CEE3SRP and CEEMRCE.*
  * PGM2 registered UCH1. This program sets a*
  * new resume point, does a divide-by-zero,*
  * and after resuming in PGM3, resets the *
  * resume point to PGM2 and does a GOBACK.*
  ________________________________________________________________
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 RECOVERY-AREA EXTERNAL.
    05 RECOVERY-POINT POINTER.
    05 ERROR-INDICATOR PIC X(01).
  01 UCH-Routine   PROCEDURE-POINTER.

Figure 142. COBOL Example of CEEMRCE (Part 2 of 4)
01 FIELDS.
 05 FIRST-TIME-SW PIC X(03) VALUE 'ON'.
 88 FIRST-TIME-88 VALUE 'ON'.
 05 ANSWER PIC S9(02) COMP-3 VALUE 0.
 05 CEEHDLR PIC X(08) VALUE 'CEEHDLR'.
 05 CEE3SRP PIC X(08) VALUE 'CEE3SRP'.
 05 TOKEN PIC S9(09) BINARY.
 05 SEV PIC -9(05).
 05 MSG PIC -9(05).
 05 FC.
    10 CASE-1.
       15 SEVERITY PIC S9(04) BINARY.
       15 MSG-NO PIC S9(04) BINARY.
       10 SEV-CTL PIC X(01).
       10 FACILITY-ID PIC X(03).
       10 I-S-INFO PIC S9(09) BINARY.

PROCEDURE DIVISION.

PERFORM COMPUTE-LOOP 3 TIMES.
SET RECOVERY-POINT TO NULL.
GOBACK.

COMPUTE-LOOP.
  IF FIRST-TIME-88
    MOVE 'OFF' TO FIRST-TIME-SW
  *__________________________________________*
  * Set new resume point.                      *
  *__________________________________________*
  * Set new resume point.                      *
  *__________________________________________*

    CALL CEE3SRP USING RECOVERY-POINT, FC
    IF CASE-1 NOT = LOW-VALUE
       GOBACK.

    IF ERROR-INDICATOR = 'E'
       MOVE SPACE TO ERROR-INDICATOR
       MOVE 1 TO ANSWER.

    * Application code may go here.

    COMPUTE ANSWER = 1 / ANSWER.

    * Put application code here.

CBL NODYNAM APOST
IDENTIFICATION DIVISION.
PROGRAM-ID. UCH1.

Figure 142. COBOL Example of CEEMRCE (Part 3 of 4)
Sample user condition handler using CEEMRCE. This program sets an error flag for the program-in-error to query and issues a call to CEEMRCE to return control to the statement following the call to CEE3SRP.

DATA DIVISION.
WORKING-STORAGE SECTION.

01 RECOVERY-AREA EXTERNAL.
   05 RECOVERY-POINT POINTER.
   05 ERROR-INDICATOR PIC X(01).

01 FC.
   10 CASE-1.
      15 SEVERITY PIC S9(04) BINARY.
      15 MSG-NO PIC S9(04) BINARY.
      10 SEV-CTL PIC X(01).
      10 FACILITY-ID PIC X(03).
      10 I-S-INFO PIC S9(09) BINARY.

01 CEEMRCE PIC X(08) VALUE 'CEEMRCE'.
LINKAGE SECTION.

01 CURRENT-CONDITION PIC X(12).
01 TOKEN PIC X(04).
01 RESULT-CODE PIC S9(09) BINARY.
   88 RESUME VALUE +10.
   88 PERCOLATE VALUE +20.
   88 PERC-SF VALUE +21.
   88 PROMOTE VALUE +30.
   88 PROMOTE-SF VALUE +31.
01 NEW-CONDITION PIC X(12).

PROCEDURE DIVISION USING CURRENT-CONDITION,
   TOKEN,
   RESULT-CODE,
   NEW-CONDITION.

   MOVE 'E' TO ERROR-INDICATOR.

   CALL CEEMRCE USING RECOVERY-POINT,
   FC.
   IF CASE-1 NOT = LOW-VALUE
   GOBACK.
   MOVE +10 TO RESULT-CODE.
   GOBACK.

Figure 142. COBOL Example of CEEMRCE (Part 4 of 4)
/* Module 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;
      Else
      Display('Second time through...');
      End;
  End;
Else
  Display('Second time through...');
/* 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.

Call CEEHDLU(PLI3SRP, FBCODE);
If FBCHECK(FBCODE, CEE000) Then
    Display('Main: unregistered PLI3SRP');
Else
    Do;
    Display('CEEHDLU failed with msg ');
    Display(MsgNo);
End;
End DRV3SRP;

*Process lc(101),opt(0),s,map,list,stmt,a(f),ag;
*Process macro;
PLI3SRP: Proc (PTR1,PTR2,PTR3,PTR4) Options(byvalue);
/******************************************************************************/
** PLI3SRP - Call CEEMRCE to resume at the resume point explicitly set in user program. */
******************************************************************************/
%include CEEIBMCT;
%include CEEIBMAW;
declare (PTR1,PTR2,PTR3,PTR4) pointer;
declare 01 CURCOND based(PTR1) feedback;
declare TOKEN pointer based(PTR2);
declare RESULT fixed bin(31,0) based(PTR3);
declare 01 NEWCOND based(PTR4) feedback;
declare 01 U_DATA based(TOKEN),
     03 U_CNTL fixed binary(31,0),
     03 U_TOK pointer;
declare 01 FBCODE feedback;

Display('In user handler');
RESULT = 10;
Call CEEMRCE(U_TOK,FBCODE);
Display(U_CNTL);
U_CNTL = 1;
Return;
End PLI3SRP;

Figure 143. PL/I Example of CEEMRCE (Part 2 of 2)
Multiple calls to CEEMRCR yield the net results of the calls; that is, if two calls move the resume cursor to different places for the same stack frame, the most restrictive call (that closest to the earliest stack frame) is used for that stack frame.

Moving the resume cursor to a particular stack frame:
- Cancels all stack frames from the previous resume point up to but not including the new resume point
- Unregisters any user condition handlers registered for the canceled stack frames

Syntax

```
CEEMRCR(type_of_move, fc)
```

**type_of_move (input)**

A fullword binary signed integer indicating the target of the resume cursor movement. The possible values for `type_of_move` are:

0  Move the resume cursor to the call return point of the stack frame associated with the handle cursor.

1  Move the resume cursor to the call return point of the stack frame prior to the stack frame associated with the handle cursor. The handle cursor is moved to the most recently established condition handler of the stack frame. The new resume cursor position now points to this condition handler for the stack frame.

Do not use a `type_of_move` value of 1 if the caller of the stack frame associated with the handle cursor is a nested COBOL program.

Modifying the resume cursor to point to stack frame 0 is not allowed. You cannot move the resume cursor beyond the earliest stack frame.

**fc (output)**

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

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 144 on page 385 routine A calls routine B, which in turn calls C, which calls D. User condition handlers are registered in routines B and C.

When a condition is raised in routine D, the Language Environment condition manager passes control to the user condition handler established for routine C. The handle cursor now points to the stack frame for routine C. Routine C percolates the condition.

The handle cursor now points to the stack frame for routine B. The next user condition handler to gain control is that one established for routine B; it recognizes the condition and issues a resume by calling CEEMRCR.

A 0 type_of_move, meaning move the resume cursor to the stack frame associated with the handle cursor, causes control to resume at the call return point in routine B, the instruction immediately following the call to routine C. A 1 type_of_move, meaning move the resume cursor to the call return point of the stack frame immediately preceding the one to which the handle cursor points, moves the resume cursor to the instruction immediately following a call in routine A to routine B.
The same scenario is illustrated in Figure 145 on page 386, except that HDLR2 issues a resume for the signaled condition rather than percolating it. HDLR1 never gains control. Because the handle cursor now points to the stack frame for routine C, a 0 type_of_move causes control to resume at the call return point in routine C, the instruction immediately following the call to routine D. A 1 type_of_move moves the resume cursor to the instruction immediately following a call in routine B to routine C.

Figure 144. First Example Moving Resume Cursor Using CEEMRCR
In Figure 146 on page 387, the user condition handlers are established for routines C and D. When a condition is raised in routine D, only a 1 type_of_move is permitted. A 0 type_of_move results in error warning message CEE0277.
Figure 146. Third Example Moving Resume Cursor Using CEEMRCR
Examples

/*Module/File Name: EDCMCR */
#include <stdio.h>
#include <string.h>
#include <leawi.h>
#include <stdlib.h>
#include <ceeedct.h>

#ifdef __cplusplus
extern "C"
{
#endif
void handler(_FEEDBACK *_,_INT4 *_,_INT4 *_,_FEEDBACK *);
#ifdef __cplusplus
}
#endif

void b(void);

int main(void) {
    /* .
     . */
    b();
    /* the CEEMCR call in the handler will place the */
    /* resume cursor at this point. */
    /* .
     . */
}

void b(void) {
    _FEEDBACK fc,condtok;
    _ENTRY routine;
    _INT4 token,qdata;
    _INT2 c_1,c_2,cond_case,sev,control;
    _CHAR3 facid;
    _INT4 isi;

    /* register the condition handler */
    token = 99;
    routine.address = (_POINTER)&handler;;
    routine.nesting = NULL;
}

Figure 147. C/C++ Example of CEEMCR (Part 1 of 2)
CEEHDLR(&routine,&token,&fc);
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEEHDLR failed with message number %d\n", fc.tok_msgno);
    exit (2999);
}

/* set up the condition using CEENCOD */
c_1 = 3;
c_2 = 2523;
cond_case = 1;
sev = 3;
control = 0;
memcpy(facid,"CEE",3);
isi = 0;
CEENCOD(&c_1,&c_2,&cond_case,&sev,&control,; facid,&isi,&condtok,&fc);
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEENCOD failed with message number %d\n", fc.tok_msgno);
    exit(2999);
}

/* signal the condition */
CEESGL(&condtok,&qdata,&fc);
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf("CEESGL failed with message number %d\n", fc.tok_msgno);
    exit(2999);
}

void handler(_FEEDBACK *fc, _INT4 *token, _INT4 *result, _FEEDBACK *newfc) {

    _FEEDBACK cursorfc, orig_fc;
    _INT4 type;
    
    /* move the resume cursor to the caller of the */
    / routine that registered the condition handler */
    type = 1;
    CEEMRCR(&type,&cursorfc);
    if ( _FBCHECK ( cursorfc , CEE000 ) != 0 ) {
        printf("CEEMRCR failed with message number %d\n", cursorfc.tok_msgno);
        exit (2999);
    }
    printf("condition handled\n");
    *result = 10;
    return;
}

Figure 147. C/C++ Example of CEEMRCR (Part 2 of 2)
CBL LIB,QUOTE
*Module/File Name: IGZTMRCR
******************************************************************************
** CBLMAIN - Main for sample program for **
** CEEMRCR. **
**
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLMAIN.
PROCEDURE DIVISION.
  CALL "DRVMRCR"
  DISPLAY "Resumed execution in the CALLER "
  "of the routine which registered the 
  "handler"
  GOBACK.
END PROGRAM CBLMAIN.
******************************************************************************
**
** DRVMRCR - Drive sample program CEEMRCR. **
**
******************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DRVMRCR.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 ROUTINE PROCEDURE-POINTER.
  01 TOKEN PIC S9(9) BINARY.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.
  01 FC.
  02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
  01 QDATA PIC S9(9) BINARY.

Figure 148. 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-Seq-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 148. 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 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 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.

Figure 148. COBOL Example of CEEMRCR (Part 3 of 4)
**Figure 148. 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.
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 <em>message-number</em> and facility ID <em>facility-id</em> could not be located.</td>
</tr>
<tr>
<td>CEE0E3</td>
<td>3</td>
<td>0451</td>
<td>An invalid destination code <em>destination-code</em> was passed to routine <em>routine-name</em>.</td>
</tr>
<tr>
<td>CEE0E6</td>
<td>3</td>
<td>0454</td>
<td>The message number <em>message-number</em> could not be found for facility ID <em>facility-id</em>.</td>
</tr>
<tr>
<td>CEE0E9</td>
<td>3</td>
<td>0457</td>
<td>The message file destination <em>ddname</em> could not be located.</td>
</tr>
<tr>
<td>CEE0EA</td>
<td>3</td>
<td>0458</td>
<td>The message repository <em>repository-name</em> 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 400](#) for more information about the CEENCOD callable service.
CEEMSG

- See ["MSGFILE" on page 47](#) for more information about the MSGFILE run-time option.

**Examples**

```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 150. C/C++ Example of CEEMSG*
CEEMSG

CBL LIB,QUOTE
*Module/File Name: IGZTMSG

*********************************************************
**
** CBLMSG - Call CEEMSG to get, format and dispatch a message
**
** In this example, CEE3MDS is called with an invalid country code so that a condition
** token would be returned to use as input to
** Any Lang Env service could have been called.
** CEEMSG uses the condition token to get, format and dispatch the message associated
** with the condition that occurred in CEE3MDS.
**
*********************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLMSG.
DATA DIVISION.
WORKING-STOREAGE 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.

Figure 151. COBOL Example of CEEMSG (Part 1 of 2)
** Call a Lang Env svc, CEE3MDS in this case, **
** to receive a condition token that CEEMSG **
** can format as a message. Specify an **
** invalid value for country code so that a **
** condition will be built **
******************************************************************************

MOVE "LN" TO COUNTRY.
CALL "CEE3MDS" USING COUNTRY, DECSEP, FC.
PARA-CBLMSG.
******************************************************************************

** Specify 2 for destination, so message will **
** be written to the ddname specified or **
** defaulted in the MSGFILE 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 151. COBOL Example of CEEMSG (Part 2 of 2)
CEEMSG

/*PROCESS LANGLEVEL(SAA), MACRO;
 /* Module/File Name: IBMMSG */
/*******************************
/**
/** Function: CEEMSG - get, format and dispatch
/** a message
/**
/** In this example, CEE3MDS is called with an
/** invalid country code so that a condition token
/** would be returned to use as input to CEEMSG.
/** Any LE/370 could have been called. CEEMSG uses
/** the condition to get, format and dispatch the
/** message associated with the condition that
/** occurred in CEE3MDS
/**
/**
/****************************/
PLIMSG: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL COUNTRY CHARACTER (2);
DCL DECSEP CHARACTER (2);
DCL 01 FCT, /* 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 152. PL/I Example of CEEMSG (Part 1 of 2)
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

```plaintext
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.

**control (input)**
A 2-byte binary integer containing flags describing or controlling various aspects of condition handling. Valid values for the control field are 1 and 0. 1 indicates the facility_ID is assigned by IBM. 0 indicates the facility_ID is assigned by the user.

**facility_ID (input)**
A 3-character field containing three alphanumeric characters (A-Z, a-z and 0-9) identifying the product or component of a product generating this condition or feedback information, for example, CEE.

The facility_ID is associated with the repository (for example, a file) of the run-time messages. If a unique ID is required (for IBM and non-IBM products), an ID can be obtained by contacting an IBM project office.

If you create a new facility_ID to use with a message file you created by using the CEEBLDTX utility, be aware that the facility_ID must be part of the message file name. It is therefore important to follow the naming guidelines described below in order to have a module name that does not cause your application to abend.

Begin a non-IBM assigned product facility_ID with letters J through Z. (See the control (input) parameter, above, on how to indicate whether the facility_ID has been assigned by IBM.) Special characters, including blank spaces, cannot be used in a facility_ID. There are no other constraints (besides the alphanumeric requirement) on a non-IBM assigned facility_ID.

**i_s_info (input)**
A fullword binary integer identifying the ISI, that contains insert data.

Whenever a condition is detected by the 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.

The layout of the type_FEEDBACK condition token in the header file is:

```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;
```

**Figure 153. type_FEEDBACK Data Type as Defined in the leawi.h Header File**

- z/OS UNIX consideration—In multithread applications, CEENCOD affects only the calling thread.

**For more information**

- For more information about case 1 and case 2, see CEENCOD “Usage notes.”
Examples

/*Module/File Name: EDCNCO */

 /*******************************************************************************/
 /* Note that it is not necessary to use this service. */
 /* The fields may be manipulated directly. */
 /*******************************************************************************/

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

 int main(void) {

   _FEEDBACK fc, condtok;
   _INT2 c_1, c_2, cond_case, sev, control;
   _CHAR3 facid;
   _INT4 isi;

   c_1 = 1;
   c_2 = 99;
   cond_case = 1;
   sev = 1;
   control = 0;
   memcpy(facid, "ZZZ", 3);
   isi = 0;

   CEENCOD(&c_1, &c_2, &cond_case, &sev, &control,
           &facid, &isi, &condtok, &fc);

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

 /* */
 /* */
 }

 Figure 154. C/C++ Example of CEENCOD
Figure 155. 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" **
*********************************************************************************************
CALC "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 155. COBOL Example of CEENCOD (Part 2 of 2)
CEENCOD

*PROCESS MACRO;
/* Module/File Name: IBMNCOD */
*************************************************************************
/** Function: CEENCOD - construct a condition token */
/** */
*************************************************************************
PLINCOD: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMFW;
%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 */
   01 MsgSev REAL FIXED BINARY(15,0),
   01 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 */
   01 MsgSev REAL FIXED BINARY(15,0),
   01 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 156. PL/I Example of CEENCOD (Part 1 of 2)
CEEQCEN—Query the century window

CEEQCEN queries the century in which Language Environment contains the 2-digit year value. When you want to change the setting, use CEEQCEN to get the setting and then use CEESCEN to save and restore the current setting.

**Syntax**

```
CEEQCEN( century_start, fc )
```

*century_start (output)*

An integer between 0 and 100 indicating the year on which the century window is based.

For example, if the Language Environment default is in effect, all 2-digit years lie within the 100-year window starting 80 years prior to the system date. CEEQCEN then returns the value 80. An 80 value indicates to Language Environment that, in 1995, all 2-digit years lie within the 100-year window starting 80 years before the system date (between 1915 and 2014, inclusive).

*fc (output)*

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

---

Figure 156. PL/I Example of CEENCOD (Part 2 of 2)
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” on page 421](#) for more information about the CEESCEN callable service.

**Examples**

```c
/*Module/File Name: EDCQCEN */

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

int main (void) {
  _INT4 century_start;
  _FEEDBACK fc;

  /* query the century window */
  CEEQCEN(&century_start,&fc);
  if (!FBCHECK (fc, CEE000) ) {
      printf("CEEQCEN failed with message number %d\n",
             fc.tok_msgno);
      exit(2999);
  }

  /* if the century window is not 50 set it to 50 */
  if (century_start != 50) {
      century_start = 50;

      CEESCEN(&century_start,&fc);
      if (!FBCHECK (fc, CEE000) ) {
          printf("CEESCEN failed with message number %d\n",
                  fc.tok_msgno);
          exit(2999);
      }
  }
}
```

*Figure 157. C/C++ Example of CEEQCEN*
CBL LIB,QUOTE
*Module/File Name: IGZTQCEN
*************************************************
** CBLQCEN - Call CEEQCEN to query the Lang Env**
** century window **
** In this example, CEEQCEN is called to query **
** the date at which the century window starts **
** The century window is the 100-year window **
** within which Lang Envir **
** assumes all two-digit years lie. **
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLQCEN.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 STARTCW PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
    COPY CEEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLQCEN.
******************************************************************************
** Call CEEQCEN to return the start of the **
** century window **
******************************************************************************
CALL "CEEQCEN" USING STARTCW, FC.
******************************************************************************
** CEEQCEN has no non-zero feedback codes to **
** check, so just display result. **
******************************************************************************
IF CEE000 of FC THEN
  DISPLAY "The start of the century "
  "window is: " STARTCW
ELSE
  DISPLAY "CEEQCEN failed with msg "
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.
GOBACK.

Figure 158. COBOL Example of CEEQCEN
CEEQDTC

**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.
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 105.

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:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>abmon</td>
<td>Abbreviated month names (12 instances of a halfword length-prefixed character string, VSTRING, of 22 bytes)</td>
</tr>
<tr>
<td>mon</td>
<td>Month names (12 instances of a halfword length-prefixed character string, VSTRING, of 62 bytes)</td>
</tr>
<tr>
<td>abday</td>
<td>Abbreviated day names (7 instances of a halfword length-prefixed character string, VSTRING, of 22 bytes)</td>
</tr>
<tr>
<td>day</td>
<td>Day names (7 instances of a halfword length-prefixed character string, VSTRING, of 62 bytes)</td>
</tr>
<tr>
<td>d_t_fmt</td>
<td>Date and time format (1 instance of a halfword length-prefixed character string, VSTRING, of 82 bytes)</td>
</tr>
<tr>
<td>d_fmt</td>
<td>Date format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)</td>
</tr>
<tr>
<td>t_fmt</td>
<td>Time format (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)</td>
</tr>
<tr>
<td>am_fmt</td>
<td>AM string (1 instance of a halfword length-prefixed character string, VSTRING, of 22 bytes)</td>
</tr>
<tr>
<td>pm_fmt</td>
<td>PM string (1 instance of a halfword length-prefixed character string, VSTRING, of 22 bytes)</td>
</tr>
<tr>
<td>t_fmt_ampm</td>
<td>Time format ampm (1 instance of a halfword length-prefixed character string, VSTRING, of 42 bytes)</td>
</tr>
</tbody>
</table>

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—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 443 for more information about the current locale.
Examples

CBL LIB,QUOTE
*Module/File Name: IGZTQDTC
**************************************************
* Example for callable service CEEQDTC  *
* MAINQDTC - Retrieve date and time convention *
*  structures for two countries and *
*   compare an item. *
* Valid only for COBOL for MVS & VM Release 2  *
* or later. *
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINQDTC.
DATA DIVISION.
WORKING-STORAGE SECTION.
  * Use DTCONV structure for CEEQDTC calls
    COPY CEEIGZDT.
  *
PROCEDURE DIVISION.
  * Subroutine needed for addressing
    CALL "COBQDTC" USING DTCONV.

STOP RUN.

* IDENTIFICATION DIVISION.
PROGRAM-ID. COBQDTC.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 Locale-Name.
    02 LN-Length PIC S9(4) BINARY.
    02 LN-String PIC X(256).
  *
  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 160. COBOL Example of CEEQDTC (Part 1 of 2)
* Call CEESETL to set all locale categories
  CALL "CEESETL" USING Locale-Name, LC-ALL, FC.

* Check feedback code
  IF Severity > 0
    DISPLAY "Call to CEESETL failed. " Msg-No
    EXIT PROGRAM
  END-IF.

* Call CEEQDTC for French values
  CALL "CEEQDTC" USING OMITTED,
  ADDRESS OF DTCONV, FC.

* Check feedback code
  IF Severity > 0
    DISPLAY "Call to CEEQDTC failed. " Msg-No
    EXIT PROGRAM
  END-IF.

* Save date and time format for FFEE locale
  MOVE D-T-FMT-Length IN DTCONV TO Test-Length1
  MOVE D-T-FMT-String IN DTCONV TO Test-String1

* Set up locale for French Canadian
  MOVE 4 TO LN-Length.
  MOVE "FCEY" TO LN-String (1:LN-Length).

* Call CEESETL to set locale for all categories
  CALL "CEESETL" USING Locale-Name, LC-ALL, FC.

* Check feedback code
  IF Severity > 0
    DISPLAY "Call to CEESETL failed. " Msg-No
    EXIT PROGRAM
  END-IF.

* Call CEEQDTC again for French Canadian values
  CALL "CEEQDTC" USING OMITTED,
  ADDRESS OF DTCONV, FC.

* Check feedback code and display results
  IF Severity = 0
* Save date and time format for FCEY locale
  MOVE D-T-FMT-Length IN DTCONV
    TO Test-Length2
  MOVE D-T-FMT-String IN DTCONV
    TO Test-String2
  IF Test-String1(1:Test-Length1) =
    Test-String2(1:Test-Length2)
    DISPLAY "Same date and time format."
  ELSE
    DISPLAY "Different formats."
    DISPLAY Test-String1(1:Test-Length1)
    DISPLAY Test-String2(1:Test-Length2)
  END-IF
  ELSE
    DISPLAY "Call to CEEQDTC failed. " Msg-No
    END-IF.

EXIT PROGRAM.
END PROGRAM COBQDTC.
*
END PROGRAM MAinqDTC.

* Figure 160. COBOL Example of CEEQDTC (Part 2 of 2)
PLIQDTC: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAM; /* 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);

Figure 161. PL/I Example of CEEQDTC (Part 1 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`.

**Syntax**

```
CEEQRYL( category, localename, fc )
```

*category (input)*

A symbolic integer number that represents all or part of the locale for a program. Depending on the value of the `localename`, these categories can be initiated by the values of global categories of corresponding names. The following values for the `category` parameter are valid:

**LC_ALL**

A 4-byte integer (with a value of -1) that affects all locale categories associated with a program’s locale.

```
/* set locale with IBM default for France */
LOCATE_NAME = 'FFEY'; /* or Fr_FR.IBM-1047 */

/* use LC_ALL category constant from CEEIBMCL */
CALL CEESETL( LOCATE_NAME, LC_ALL, FC );

/* retrieve date and time structure, France Locale */
CALL CEEQDTC( *, LOCALDT, FC );

/* set locale with French Canadian(FCEY) defaults */
/* literal constant -1 used to set all categories */
CALL CEESETL( 'FCEY', -1, FC );

/* retrieve date and time tables for French Canada */
/* example of temp pointer used for service call */
CALL CEEQDTC( *, ADDR(DTCONVC), FC );

/* compare date and time formats for two countries */
IF DTCONVC.D_T_FMT = DTCONV.D_T_FMT THEN
  DO;
    PUT SKIP LIST('Countries have same D_T_FMT' );
  END;
ELSE
  DO;
    PUT SKIP LIST('Date and Time Format ',
                  DTCONVC.D_T_FMT || ' vs ' ||
                  DTCONV.D_T_FMT );
  END;
END PLIQDTC;
```

Figure 161. PL/I Example of CEEQDTC (Part 2 of 2)
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.

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 443 for more information about the current locale.
- For more information about LC_TIME, see "CEEQDTC—Query locale date and time conventions" on page 410.
- For information on the definition of the SAA C and POSIX C locales and the differences between them, see z/OS C/C++ Programming Guide.

Examples
For examples of how to use CEEQRYL in combination with other Language Environment locale callable services, see "CEESETL—Set locale operating environment" on page 443.

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

```
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

```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 162. C/C++ Example of CEERAN0
CBL Lib,Quote
*Module/File Name: IGZTRAN0
**************************************************************************
** CBLRAN0 - Call CEERAN0 to generate uniform **
** random numbers **
**************************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLRAN0.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SEED PIC S9(9) BINARY.
01 RANDNUM COMP-2.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
         04 Severity PIC S9(4) BINARY.
         04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
         REDEFINES Case-1-Condition-ID.
         04 Class-Code PIC S9(4) BINARY.
         04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
   02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLRAN0.
**************************************************************************
** Specify 0 for SEED, so the seed will be **
** derived from the current Greenwich Mean Time **
**************************************************************************
   MOVE 0 TO SEED.
**************************************************************************
** Call CEERAN0 to return random number between **
** 0.0 and 1.0 **
**************************************************************************
   CALL "CEERAN0" USING SEED, RANDNUM, FC.
**************************************************************************
** If CEERAN0 runs successfully, display result. **
**************************************************************************
   IF CEE000 of FC THEN
      DISPLAY "The random number is: " RANDNUM
   ELSE
      DISPLAY "CEERAN0 failed with msg "
      Msg-No of FC UPON CONSOLE
      STOP RUN
   END-IF.
   GOBACK.

Figure 163. COBOL Example of CEERAN0
CEESCEN—Set the century window

CEESCEN sets the century in which Language Environment contains the 2-digit year value. Use it in conjunction with CEEDAYS or CEESECS when:

- You process date values containing 2-digit years (for example, in the YYMMDD format).
- The Language Environment default century interval does not meet the requirements of a particular application.

Century intervals are kept as thread-level data, so changing the interval in one thread does not affect the interval in another thread. To query the century window, use CEEQCEN.

---

*PROCESS MACRO;
/* Module/File Name: IBMRAN0    */
/*******************-----------------------------------------------------*/
/**    */
/** Function: CEERAN0 - calculate uniform random    */
/**    */
/**    */
/*******************-----------------------------------------------------*/
PLIRAN0: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL SEED REAL FIXED BINARY(31,0);
DCL RANDNUM REAL FLOAT DECIMAL(16);
DCL 01 FC,    /* Feedback token */
    03 MsgSev REAL FIXED BINARY(15,0),
    03 MsgNo REAL FIXED BINARY(15,0),
    03 Flags,
        05 Case BIT(2),
        05 Severity BIT(3),
        05 Control BIT(3),
    03 FacID CHAR(3),    /* Facility ID */
    03 ISI /* Instance-Specific Information */
        REAL FIXED BINARY(31,0);

SEED = 7;    /* Specify an integer as the initial    */
            /* value used to calculate the random    */
            /* numbers    */

/* Call CEERAN0 to generate random number between    */
/* 0.0 and 1.0    */
CALL CEERAN0 ( SEED, RANDNUM, FC );

IF FBCHECK( FC, CEE000) THEN DO;
    PUT SKIP LIST ('The random number is ' || RANDNUM );
    END;
ELSE DO;
    DISPLAY( 'CEERAN0 failed with msg ' || FC.MsgNo );
    STOP;
    END;

END PLIRAN0;

---

Figure 164. PL/I Example of CEERAN0
CEESCEN

CEESCEN affects and is affected by only the Language Environment NLS and date and time services, not the Language Environment locale callable services or the C locale-sensitive functions.

Syntax

```
 CEERAN0((century_start, fc))
```

century_start

An integer between 0 and 100, setting the century window.

A value of 80, for example, places all two-digit years within the 100-year window starting 80 years before the system date. In 1995, therefore, all two-digit years are assumed to represent dates between 1915 and 2014, inclusive.

fc (output)

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

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 242 for more information about the CEEDAYS callable service.
- See “CEESECS—Convert timestamp to seconds” on page 435 for more information about the CEESECS callable service.
- See “CEEQCEN—Query the century window” on page 407 for more information about the CEEQCEN callable service.
Examples

/*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 165. C/C++ Example of CEESCEN
CBL LIB,QUOTE
*Module/File Name: IGZTSCEN
**************************************************
** CBLSCEN - Call CEESCEN to set the Lang. Env. **
** century window **
**
** In this example, CEESCEN is called to change **
** the start of the century window to 30 years **
** before the system date. CEEQCN is then **
** called to query that the change made. A **
** message that this has been done is then **
** displayed. **
**
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLSCEN.

DATA DIVISION.
WORKING-STORAGE SECTION.
 01 STARTCW PIC S9(9) BINARY.
 01 FC.
    02 Condition-Token-Value.
      COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
PARA-CBLSCEN.

**************************************************
** Specify 30 as century start, and two-digit **
** years will be assumed to lie in the **
** 100-year window starting 30 years before **
** the system date. **
**************************************************
MOVE 30 TO STARTCW.

**************************************************
** Call CEESCEN to change the start of the century **
** window. **
**************************************************

Figure 166. COBOL Example of CEESCEN (Part 1 of 2)
CALL "CEESCEN" USING STARTCW, FC.
IF NOT CEE000 of FC THEN
    DISPLAY "CEESCEN failed with msg "
    Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.
PARA-CBLQCEN.
******************************************************************************
** Call CEEQCEN to return the start of the century
** window
******************************************************************************
CALL "CEEQCEN" USING STARTCW, FC.
******************************************************************************
** CEEQCEN has no non-zero feedback codes to check, so just display result.
******************************************************************************
DISPLAY "The start of the century "
    "window is: " STARTCW
GOBACK.

Figure 166. COBOL Example of CEESCEN (Part 2 of 2)
CEESCOL—Compare collation weight of two strings

CEESCOL, analogous to the C language function `strcoll()`, compares two character strings based on the collating sequence specified in the LC_COLLATE category of the current locale.

CEESCOL associates a collation weight with every character in the code set for the locale. The characters in the input strings are converted to their collation weights and then compared on the basis of these weights.

CEESCOL is sensitive to the locales set by `setlocale()` or `CEESETL`, not to the Language Environment settings from `COUNTRY` or `CEE3CTY`.

```
*PROCESS MACRO;
/* Module/File Name: IBMSCEN */
/**************************************************************/
/** Function: CEESCEN - set the century window **/
/** **************************************************************/
PLISCEN: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMMAW;
%INCLUDE CEEIBMCT;

DCL STARTCW REAL FIXED BINARY(31,0);
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);

STARTCW = 20; /* Set 20 as century start */

/* Call CEESCEN to request that two-digit years */
/* lie in the 100-year window starting 20 */
/* years before the system date */
CALL CEESCEN (STARTCW, FC);
IF FBCHECK( FC, CEE000) THEN DO;
  PUT SKIP LIST ('The century window now starts '
    || STARTCW || ' years before today.');
END;
ELSE DO;
  DISPLAY( 'CEESCEN failed with msg ' || FC.MsgNo );
  STOP;
END;

END PLISCEN;
```

Figure 167. PL/I Example of CEESCEN
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 105.

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>4001</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 443 for more information about the CEESETL callable service.
Examples

CBL LIB,QUOTE
*Module/File Name: IGZTSCOL
*********************************************************************
* Example for callable service CEESCOL     *
* COBSCOL - Compare two character strings     *
*   and print the result.     *
* Valid only for COBOL for MVS & VM Release 2     *
* or later.     *
*********************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSCOL.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 String1.
   02 Str1-Length PIC S9(4) BINARY.
   02 Str1-String.
      03 Str1-Char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Str1-Length.
01 String2.
   02 Str2-Length PIC S9(4) BINARY.
   02 Str2-String.
      03 Str2-Char PIC X OCCURS 0 TO 256 TIMES DEPENDING ON Str2-Length.
01 Result PIC S9(9) BINARY.
01 FC.
   02 Condition-Token-Value.
      COPY CEEIGZCT.
         03 Case-1-Condition-ID.
            04 Severity PIC S9(4) BINARY.
            04 Msg-No PIC S9(4) BINARY.
            03 Case-2-Condition-ID
               REDEFINES Case-1-Condition-ID.
               04 Class-Code PIC S9(4) BINARY.
               04 Cause-Code PIC S9(4) BINARY.
               03 Case-Sev-Ctl PIC X.
               03 Facility-ID PIC XXX.
               02 1-S-Info PIC S9(9) BINARY.
   * PROCEDURE DIVISION.
*********************************************************************
* Set up two strings for comparison
*********************************************************************
   MOVE 9 TO Str1-Length.
   MOVE "12345a789" TO Str1-String (1:Str1-Length)
   MOVE 9 TO Str2-Length.
   MOVE "12346$789" TO Str2-String (1:Str2-Length)

Figure 168. 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.

Figure 168. COBOL Example of CEESCOL (Part 2 of 2)
CEESECI—Convert seconds to integers

CEESECI converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. Use CEESECI instead of CEEDATM when the output is needed in numeric format rather than in character format.
The inverse of CEESECI is CEEISEC, which converts integer year, month, day, hour, second, and millisecond to number of seconds. If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86,400 (number of seconds in a day), and pass the new value to CEESECI.

Syntax

```
CEESECI(input_seconds, output_year, output_month, output_day, output_hours, output_minutes, output_seconds, output_milliseconds, fc)
```

input_seconds
A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582, not counting leap seconds.

For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range for input_seconds is 86,400 to 265,621,679,999.999 (23:59:59.999 31 December 9999).

If input_seconds is invalid, all output parameters except the feedback code are set to 0.

output_year (output)
A 32-bit binary integer representing the year.

The range of valid output_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.
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 341.

**Examples**

```c
/*Module/File Name: EDCSECI */

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

int main(void) {
   _INT4 year, month, day, hours, minutes, seconds, millisecs;
   _FLOAT8 input;
   _FEEDBACK fc;

   input = 13166064000.0;
   CEESECI(&input,&year,&month,&day,&hours,&minutes;,
       &seconds,&millisecs,&fc);

   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf("CEESECI failed with message number %d\n", fc.tok_msgno);
      exit(2999); 
   }
   printf("%f seconds corresponds to the date" 
   " %d:%d:%d %d/%d/%d\n",input,hours,minutes, 
       seconds,millisecs,month,day,year); 
}
```

*Figure 170. C/C++ Example of CEESECI*
**Module/File Name: IGZTSECI
***********************************************************************************************************************************************
**
** CBLSECI - Call CEESECI to convert seconds **
** to integers **
**
** In this example a call is made to CEESECI **
** to convert a number representing the number **
** of seconds since 00:00:00 14 October 1582 **
** to seven binary integers representing year, **
** month, day, hour, minute, second, and **
** millisecond. The results are displayed in **
** this example. **
**
***********************************************************************************************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. CBLSECI.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 INSECS  COMP-2.
01 YEAR    PIC S9(9) BINARY.
01 MONTH   PIC S9(9) BINARY.
01 DAYS    PIC S9(9) BINARY.
01 HOURS   PIC S9(9) BINARY.
01 MINUTES PIC S9(9) BINARY.
01 SECONDS PIC S9(9) BINARY.
01 MILSEC  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-Dev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
PROCEDURE DIVISION.
PARA-CBLSECS.

Figure 171. 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.

PARA-CBLSECI.

Call CEESECI to convert seconds to integers

CALL "CEESECI" USING INSECS, YEAR, MONTH, DAYS, HOURS, MINUTES, SECONDS, MILLSEC, FC.

If CEESECI runs successfully, display results

IF CEE000 of FC THEN
   DISPLAY "Input seconds of " INSECS " represents:"
   DISPLAY " Year........ " YEAR
   DISPLAY " Month........ " MONTH
   DISPLAY " Day........... " DAYS
   DISPLAY " Hour......... " HOURS
   DISPLAY " Minute....... " MINUTES
   DISPLAY " Second....... " SECONDS
   DISPLAY " Millisecond.. " MILLSEC
ELSE
   DISPLAY "CEESECI failed with msg "
   Msg-No of FC UPON CONSOLE
   STOP RUN
END-IF.

GOBACK.

Figure 171. COBOL Example of CEESECI (Part 2 of 2)
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

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 CEESCEN.

**Syntax**

```
| CEESECS(input_timestamp, picture_string, output_seconds, fc) |
```

**input_timestamp (input)**
A length-prefixed character string representing a date or timestamp in a format matching that specified by picture_string.

The character string must contain between 5 and 80 picture characters, inclusive. input_timestamp can contain leading or trailing blanks. Parsing begins with the first nonblank character (unless the picture string itself contains leading blanks; in this case, CEESECS skips exactly that many positions before parsing begins).

After a valid date is parsed, as determined by the format of the date you specify in picture_string, all remaining characters are ignored by CEESECS. Valid dates range between and including the dates 15 October 1582 to 31 December 9999. A full date must be specified. Valid times range from 00:00:00.000 to 23:59:59.999.

If any part or all of the time value is omitted, zeros are substituted for the remaining values. For example:

```
1992-05-17-19:02 is equivalent to 1992-05-17-19:02:00
1992-05-17 is equivalent to 1992-05-17-00:00:00
```

**picture_string (input)**
A halfword length-prefixed character string (VSTRING), indicating the format of the date or timestamp value specified in input_timestamp.

Each character in the picture_string represents a character in input_timestamp. For example, if you specify MMDDYY HH.MM.SS as the picture_string, CEESECS reads an input_char_date of 060288 15.35.02 as 3:35:02 PM on 02 June 1988. If delimiters such as the slash (/) appear in the picture string, leading zeros can be omitted. For example, the following calls to CEESECS all assign the same value to variable secs:

```
CALL CEESECS('92/06/03 15.35.03', 'YY/MM/DD HH.MM.SS', secs, fc);
CALL CEESECS('92/6/3 15.35.03', 'YY/MM/DD HH.MM.SS', secs, fc);
CALL CEESECS('92/6/3 3.35.03 PM', 'YY/MM/DD HH.MM.SS AP', secs, fc);
CALL CEESECS('92.155 3.35.03 pm', 'YY.DDD HH.MM.SS AP', secs, fc);
```
If picture string is left null or blank, CEESECS gets picture_string based on the current value of the COUNTRY run-time option. For example, if the current value of the COUNTRY run-time option is FR (France), the date format would be DD.MM.YYYY.

If picture_string includes a Japanese era symbol <JJJJ>, the YY position in input_timestamp represents the year number within the Japanese era. For example, the year 1988 equals the Japanese year 63 in the Showa era. See Table 31 on page 520 for a list of Japanese eras supported by CEESECS.

If picture_string includes era symbol <CCCC> or <CCCCCCC>, the YY position in input_timestamp represents the year number within the era. See Table 29 on page 519 for a list of valid picture characters, and Table 30 on page 520 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>
</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 421 for more information about the CEESCEN callable service.
- See "COUNTRY" on page 22 for more information about the COUNTRY run-time option.
Examples

/*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
",
                fc.tok_msgno);
        exit(2999);
    }

    date.string = "December 15, 1992 at 8:23:45 AM";
    date.length = strlen(date.string);
    date_pic.string = "Mmmmmmmmmmm 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
",
                fc.tok_msgno);
        exit(2999);
    }

    printf("The number of seconds between:
");
    printf(" September 13, 1991 at 11:23:23 PM\n");
    printf(" and December 15, 1992 at 8:23:45 AM is: \n%f\n", seconds2 - seconds1);
}

Figure 173. C/C++ Example of CEESECS
CBL LIB,QUOTE

*Module/File Name: IGZTSECS
************************************************
** CBLSECS - Call CEESECS to convert **
** timestamp to number of seconds **
**
** In this example, calls are made to CEESECS **
** to convert two timestamps to the number of **
** seconds since 00:00:00 14 October 1582. **
** The Lilian seconds for the earlier **
** timestamp are then subtracted from the **
** Lilian seconds for the later timestamp **
** to determine the number of between the **
** two. This result is displayed. **
************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLSECS.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 SECOND1   COMP-2.  
01 SECOND2   COMP-2.  
01 TIMESTP.  
   02 Vstring-length PIC S9(4) BINARY.  
   02 Vstring-text.  
      03 Vstring-char PIC X, OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of TIMESTP.  
01 TIMESTP2.  
   02 Vstring-length PIC S9(4) BINARY.  
   02 Vstring-text.  
      03 Vstring-char PIC X, OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of TIMESTP2.  
01 PICSTR.  
   02 Vstring-length PIC S9(4) BINARY.  
   02 Vstring-text.  
      03 Vstring-char PIC X, OCCURS 0 TO 256 TIMES DEPENDING ON Vstring-length of PICSTR.  
01 FC.  
   02 Condition-Token-Value.

Figure 174. COBOL Example of CEESECS (Part 1 of 3)
COPY CEEIGZCT.
  03 Case-1-Condition-ID.
    04 Severity PIC S9(4) BINARY.
    04 Msg-No PIC S9(4) BINARY.
  03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
    04 Class-Code PIC S9(4) BINARY.
    04 Cause-Code PIC S9(4) BINARY.
  03 Case-Sev-Ctl PIC X.
  03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) BINARY.

PROCEDURE DIVISION.
  PARA-SECS1.

************************************************
** Specify first timestamp and a picture string
** describing the format of the timestamp
** as input to CEESECS
************************************************
MOVE 25 TO Vstring-length of TIMESTP.
MOVE "1969-05-07 12:01:00.000" TO Vstring-text of TIMESTP.
MOVE 25 TO Vstring-length of PICSTR.
MOVE "YYYY-MM-DD HH:MI:SS.999" TO Vstring-text of PICSTR.

************************************************
** Call CEESECS to convert the first timestamp
** to Lilian seconds
************************************************
CALL "CEESECS" USING TIMESTP, PICSTR, SECOND1, FC.
IF NOT CEE000 of FC THEN
  DISPLAY "CEESECS failed with msg "
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

Figure 174. COBOL Example of CEESECS (Part 2 of 3)
** 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.  

** 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.
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.
CEESETL

Syntax

```c
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>
<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 CEEQRYL returns “C” as the locale name.
  - With POSIX(ON), the POSIX C locale is chosen, and querying the locale with CEEQRYL returns “POSIX” as the locale name.

The SAA C locale provides compatibility with previous releases of C/370. The POSIX C locale provides consistency with POSIX requirements and supports the z/OS UNIX environment.

For more information

- See [z/OS 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 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 C/C++ Programming Guide](#).
- For more information about LC_TIME, see "CEEQDTC—Query locale date and time conventions" on page 410.
Examples

```cobol
CBL LIB,QUOTE
+Module/File Name: IGZTSETL
*************************************************
* Example for callable service CEESETL *
* COBSETL - Set all global locale environment *
* categories to country Sweden. *
* Query one category. *
*************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSETL.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 Locale-Name.
    02 LN-Length PIC S9(4) BINARY.
    02 LN-String PIC X(256).
  01 Locale-Time.
    02 LT-Length PIC S9(4) BINARY.
    02 LT-String PIC X(256).
* Use Locale category constants
COPY CEEIGZLC.
* 01 FC.
    02 Condition-Token-Value.
COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
      02 I-S-Info PIC S9(9) BINARY.
*   
```

Figure 176. COBOL Example of CEESETL (Part 1 of 2)
PROCEDURE DIVISION.
*************************************************
* Set up locale name for Sweden
*************************************************
MOVE 14 TO LN-Length.
MOVE 'Sv_SE.IBM-1047' TO LN-String (1:LN-Length).
*************************************************
* Set all locale categories to Sweden
* Use LC-ALL category constant from CEEIGZLC
*************************************************
CALL 'CEESETL' USING Locale-Name, LC-ALL, FC.

*************************************************
* Check feedback code
*************************************************
IF Severity > 0
   DISPLAY 'Call to CEESETL failed. ' Msg-No
   STOP RUN
END-IF.

*************************************************
* Retrieve active locale for LC-TIME category
*************************************************
CALL 'CEEQRYL' USING LC-TIME, Locale-Time, FC.

*************************************************
* Check feedback code and correct locale
*************************************************
IF Severity = 0
   IF LT-String(1:LT-Length) = LN-String(1:LN-Length)
      DISPLAY 'Successful query.'
   ELSE
      DISPLAY 'Unsuccessful query.'
      END-IF
ELSE
   DISPLAY 'Call to CEEQRYL failed. ' Msg-No
END-IF
STOP RUN.
END PROGRAM COBSETL.

Figure 176. COBOL Example of CEESETL (Part 2 of 2)
CEESETL

*PROCESS MACRO;
/*Module/File Name: IBMSETL */
/****************************************************************************/
/* Example for callable service CEESETL */
/* Function: Set all global locale environment */
/* categories to country. Query one category. */
/****************************************************************************/

PLISETL: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW; /* ENTRY defs, macro defs */
%INCLUDE CEEIBMCT; /* FBCHECK macro, FB constants */
%INCLUDE CEEIBMLC; /* Locale category constants */

/* CEESETL service call arguments */
DCL LOCALE_NAME CHAR(14) VARYING;

/* CEEQRYL service call arguments */
DCL LOCALE_NAME_TIME CHAR(256) VARYING;

DCL 01 FC,          /* Feedback token */
  03 MsgSev   REAL FIXED BINARY(15,0),
  03 MsgNo    REAL FIXED BINARY(15,0),
  03 Flags,   /*
    05 Case     BIT(2),
    05 Severity  BIT(3),
    05 Control  BIT(3),
    */
  03 FacID    CHAR(3),  /* Facility ID */
  03 ISI      /* Instance-Specific Information */
               REAL FIXED BINARY(31,0);

Figure 177. COBOL Example of CEESETL (Part 1 of 2)
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 via CEEHDLR. Conditions can also be selected to simulate a Language Environment or system condition. If you plan on using a routine that signals a new condition with a call to CEESGL, you should first call the CEECMI callable service to copy any insert information into the ISI associated with the condition.

CEESGL generates a Language Environment condition. You can map some of the Language Environment condition tokens to POSIX signals.

Unique conditions signaled by CEESGL are considered to be enabled under Language Environment. Therefore, they undergo Language Environment condition handling.
CEESGL can signal a POSIX condition. If CEESGL signals a POSIX condition and the signal is blocked at the time of the generation but later unblocked and delivered, the POSIX signal processing semantics are applied. The Language Environment synchronous condition manager semantics do not apply.

Severity 0 and 1 conditions are considered safe conditions. They can be ignored if they are not handled and if no feedback token is passed when the condition is raised.

Each signaled condition (of severity 2 or above) increments the error count by one. If the error count exceeds the error count limit (as specified by the ERRCOUNT run-time option—see "ERRCOUNT" on page 28) 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 192 contains a list of the S/370 program interrupt codes and their corresponding Language Environment condition token names and message numbers.

### Syntax

```plaintext
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 whether 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.

  q_data tokens associated with a condition using CEESGL can be extracted later using the CEEGQDT callable 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>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.

For more information

- See “CEECMI—Store and load message insert data” on page 208 for more information about the CEECMI callable service.
- See “ERRCOUNT” on page 28 for more information about the ERRCOUNT run-time option.
- To construct your own condition token, see “CEENCOD—Construct a condition token” on page 400.
- For more information about condition handling, see [z/OS Language Environment Programming Guide](https://www.ibm.com/support/docview.wss?uid=swg21544754).
- See “CEEGQDT—Retrieve q_data_token” on page 309 for more information about the CEEGQDT callable service.
Examples

/*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 178. C/C++ Example of CEESGL
CBL LIB,QUOTE
*Module/File Name: IGZTSGL
**************************************************
** CBLSGL - Call CEESGL to signal a condition **
**************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLSGL.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 CONDTOK.
  02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
  01 QDATA PIC S9(9) BINARY.
  01 FC.
    02 Condition-Token-Value.
    COPY CEEIGZCT.
      03 Case-1-Condition-ID.
        04 Severity PIC S9(4) BINARY.
        04 Msg-No PIC S9(4) BINARY.
      03 Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04 Class-Code PIC S9(4) BINARY.
        04 Cause-Code PIC S9(4) BINARY.
      03 Case-Sev-Ctl PIC X.
      03 Facility-ID PIC XXX.
    02 I-S-Info PIC S9(9) BINARY.
  01 SEV PIC S9(4) BINARY.
  01 MSGNO PIC S9(4) BINARY.
  01 CASE PIC S9(4) BINARY.
  01 SEV2 PIC S9(4) BINARY.
  01 CNTRL PIC S9(4) BINARY.
  01 FACID PIC X(3).
  01 ISINFO PIC S9(9) BINARY.

PROCEDURE DIVISION.

Figure 179. 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 179. COBOL Example of CEESGL (Part 2 of 2)
CEESTXF—Transform string characters into collation weights

CEESTXF, 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.
CEESTXF

Syntax

```
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 105.

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-prefix 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 120, and "CEE3LNG—Set national language" on page 163.
- For more information about the CEESETL callable service, see "CEESETL—Set locale operating environment" on page 443.
Examples

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
        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-3047"
    MOVE 10 TO MBS-Length
    MOVE 10 TO MBS-Size
  MOVE "7,123,456."
  TO MBS-String (1:MBS-Length)

Figure 181. COBOL Example of CEESTXF (Part 1 of 2)
ELSE
  MOVE 7 TO MBS-Length
  MOVE 7 TO MBS-Size
  MOVE "8765432" TO MBS-String (1:MBS-Length)
END-IF
ELSE
  DISPLAY "Call to CEEQRYL failed. " Msg-No
  STOP RUN
END-IF.
MOVE SPACES TO TXF-String.
MOVE 0 to TXF-Length.

*************************************************
  * Call CEESTXF to translate the string
  CALL "CEESTXF" USING OMITTED, MBS, MBS-Size, TXF, TXF-Size, FC.
  *************************************************

  * Check feedback code and return length
  IF Severity = 0
    IF TXF-Length > 0
      DISPLAY "Translated string is "
      TXF-String
    ELSE
      DISPLAY "String not translated."
    END-IF
  ELSE
    DISPLAY "Call to CEESTXF failed. " Msg-No
  END-IF.
  STOP RUN.
END PROGRAM COBSTXF.

Figure 181. COBOL Example of CEESTXF (Part 2 of 2)

*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 182. PL/I Example of CEESTXF (Part 1 of 2)
CEETDLI—Invoke IMS

CEETDLI provides an interface to PL/I facilities that operates in IMS. In assembler, COBOL, PL/I, and C/C++, you can also invoke IMS by using the following interfaces:

- In assembler, the ASMTDLI interface
- In COBOL, the CBLTDLI interface
- In PL/I, the PLITDLI interface

```plaintext
/* CEESTXF service call arguments */
DCL MBSTRING CHAR(10) VARYING; /* input string */
DCL MBNUMBER BIN FIXED(31); /* input length */
DCL TXFSTRING CHAR(256) VARYING; /* output string */
DCL TXFLENGTH BIN FIXED(31); /* output length */

/* retrieve active locale for collate category */
/* Use LC_COLLATE category const from CEEIBMCL */
CALL CEEQRYL (LC_COLLATE, LOCALE_NAME_COLLATE, FC);
/* FBCHECK macro used (defined in CEEIBMCT) */
IF FBCHECK( FC, CEE000 ) THEN
  DO; /* successful query, set string for CEESTXF */
    IF LOCALE_NAME_COLLATE = 'Sv_SE.IBM-1047' THEN
      MBSTRING = '7,123,456.';
    ELSE
      MBSTRING = '8765432';
      MBNUMBER = LENGTH(MBSTRING);
    END;
  ELSE
    DISPLAY ( 'Locale LC_COLLATE ' || FC.MsgNo );
    STOP;
  END;
  TXFSTRING = ';
  CALL CEESTXF ( *, MBSTRING, MBNUMBER,
                     TXFSTRING, TXFLENGTH, FC );
  IF FBCHECK( FC, CEE000 ) THEN
    DO; /* successful call, use transformed length */
      IF TXFLENGTH > 0 THEN
        DO;
          PUT SKIP LIST( 'Transformed string is ' ||
                         SUBSTR(TXFSTRING,1, TXFLENGTH) );
        END;
      END;
    ELSE
      IF FBCHECK( FC, CEE3TF ) THEN
        DO;
          DISPLAY ( 'Zero length input string' );
        END;
      END;
    END PLISTXF;
```

Figure 182. PL/I Example of CEESTXF (Part 2 of 2)
CEETDLI

- 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

```plaintext
CEETDLI( parmcount, function, args )
```

**parmcount**
A fullword integer specifying the total number of arguments for the CEETDLI call. This option usually is not needed and can be omitted; it is supported for compatibility with earlier interface modules.

**function**
The IMS function that you want to perform. The possible values for this field are defined by IMS, not Language Environment.

**args**
Arguments that you pass to IMS. You cannot pass 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 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

/*Module/FILE Name: EDCMRCR */

#pragma runopts(env(IMS),plist(IMS))
#include <ims.h>
#include <leawi.h>
#define io_pcb ((IO_PCB_TYPE *)__pcblist??(0??))

/* ------------------------------- */
/* Function: Use CEETDLI - interface to IMS */
/* from C. */
/* In this example, a call is made to CEETDLI */
/* to interface to IMS for an IMS service. */
/* ------------------------------- */
/* ENTRY POINT */
/* ------------------------------- */

main()
{
    static char func_GU??(4??) = "GU ";
    char msg_seg_io_area??(32??);
    CEETDLI(func_GU,io_pcb,msg_seg_io_area);
}

Figure 183. C Example of CEETDLI

*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. CBLZIMS.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 FUNC-GU   PIC X(4) VALUE "GU ".
  01 MSG-SEG-IO-AREA   PIC X(32).
LINKAGE SECTION.
  01 IO-PCB   PIC X(48).
  ** ENTRY POINT: */
PROCEDURE DIVISION USING IO-PCB.
    CALL "CEETDLI" USING FUNC-GU,IO-PCB,
          MSG-SEG-IO-AREA.
    GOBACK.

Figure 184. COBOL Example of CEETDLI
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 restrictions noted in [Debug Tool Users Guide](#). 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:

<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>CEE2F2</td>
<td>3</td>
<td>2530</td>
<td>A debug tool was not available.</td>
</tr>
<tr>
<td>CEE2F7</td>
<td>3</td>
<td>2535</td>
<td>Profiler loaded; a debug tool was not available.</td>
</tr>
</tbody>
</table>

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 [Debug Tool Users Guide](#).
Examples

/ * Module/File Name: EDCTEST */

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

int main (void) {

int x,y,z;
_VSTRING commands;
_FEEDBACK fc;

strcpy(commands.string, "AT LINE 30 { LIST(x); LIST(y); GO; }");
commands.length = strlen(commands.string);

CEETEST(&commands,&fc);

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

x = y = 12;
/* .
   */
/* debug tool displays the values of x and y */
/* at statement 30 */
/* .
   . */
}

Figure 186. C/C++ Example of CEETEST
CBL LIB,QUOTE
*Module/File Name: IGZTTEST
IDENTIFICATION DIVISION.
PROGRAM-ID. IBCT002.

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 MANVAR1 PIC 9(9) BINARY.
  01 CEETEST-PARMS.
   02 Vstring-length PIC 9(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 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.
PARA-IBCT002.
  MOVE 0 TO MANVAR1
  COMPUTE MANVAR1 = MANVAR1 + 100
  DISPLAY "The value of MANVAR1 is " , MANVAR1
*** CALL CEETEST FOR FIRST TIME. ***
*******************************************************************************
-- CALL CEETEST FOR FIRST TIME. --
*******************************************************************************
  MOVE 70 TO Vstring-length of CEETEST-PARMS.
  MOVE "DESC PROGRAM AT ENTRY IBCT002:>SUBRTN" TO Vstring-text of CEETEST-PARMS(1:37).
  move "PERFORM Q LOC GO END-PERFORM GO " TO Vstring-text of CEETEST-PARMS(38:33).
  CALL "CEETEST" USING CEETEST-PARMS, FC.
  IF NOT CEE000 of FC THEN
     DISPLAY "CEETEST(1st call) failed with msg "
     Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

Figure 187. 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.
**************************************************
GOBACK.

IDENTIFICATION DIVISION.
PROGRAM-ID. SUBRTN.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 MANVAR1           PIC S9(9) BINARY.
  01 MVAR               PIC S9(9) BINARY.
PROCEDURE DIVISION.
 PARA-SUBRTN.
  COMPUTE MVAR = MVAR + 100 .
  COMPUTE MANVAR1 = MANVAR1 + 100 .
  GOBACK.
END PROGRAM SUBRTN.
END PROGRAM IBCT002.

Figure 187. COBOL Example of CEETEST (Part 2 of 2)
CEEUTC

CEEUTC—Get Coordinated Universal Time

CEEUTC is an alias of CEEGMT. See “CEEGMT—Get current Greenwich Mean Time” on page 296.

Figure 188. PL/I Example of CEETEST

```
*PROCESS MACRO;
/* Module/File Name: IBMTEST */
/*****************************************************************************/
/** */
/** Function: CEETEST - Invoke a Debug Tool /**
/** */
/*****************************************************************************/
PLITEST: PROC OPTIONS(MAIN);

%INCLUDE CEEIBMAW;
%INCLUDE CEEIBMCT;

DCL DBGCMD CHAR(255) VARYING;
DCL 01 FC, /* Feedback token */
   03 MsgSev REAL FIXED BINARY(15,0),
   03 MsgNo REAL FIXED BINARY(15,0),
   03 Flags,
      05 Case BIT(2),
      05 Severity BIT(3),
      05 Control BIT(3),
   03 FacID CHAR(3), /* Facility ID */
   03 ISI /* Instance-Specific Information */
      REAL FIXED BINARY(31,0);

DBGCMD = 'QUERY PROGRAMMING LANGUAGE';

CALL CEETEST (DBGCMD, FC);
IF FBCHECK( FC, CEE000) THEN DO;
   PUT SKIP LIST('Debug tool called with command: '
      || DBGCMD );
   END;
ELSE DO;
   DISPLAY('CEETEST failed with msg ' || FC.MsgNo);
   STOP;
   END;
END PLITEST;
```
CEEUTC
Chapter 5. Bit manipulation routines

This section contains an alphabetical list of the Language Environment bit manipulation routines.

In this section, bits are numbered from right to left, starting from 0.

**CEESICLR—Bit clear**

CEESICLR returns a copy of its `parm1` input, but with a single bit selectively set to 0.

**Syntax**

```
parity

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 input is a 32-bit integer in the range $0 \leq \text{parm2} \leq 31$.

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

  The following symbolic conditions can result from this service:

- **result** (output)
  The result of the Bit Clear routine. The output is a copy of `parm1`, but with the bit numbered `parm2` (counting from the right) set to 0.

**CEESISET—Bit set**

CEESISET returns a copy of its `parm1` input, but with a single bit selectively set to 1.

**Syntax**

```
parity

CEESISET( parm1, parm2, fc, result )
```

- **parm1** (input)
  The first input to the Bit Set routine. The input can be any 32-bit integer.

- **parm2** (input)
  The second input to the Bit Set routine. The input is a 32-bit integer in the range $0 \leq \text{parm2} \leq 31$.

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

The following symbolic conditions can result from this service:

result (output)
The result of the Bit Set routine. The output is a copy of parm1, but with the bit numbered parm2 (counting from the right) set to 1.

CEESISHF—Bit shift

CEESISHF returns a copy of its parm1 input right- or left-shifted by the number of bits indicated by parm2.

Syntax

```plaintext
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 input is a 32-bit integer in the range \(-32 \leq \text{parm2} \leq 32\).

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

The following symbolic conditions can result from this service:

result (output)
The result of the Bit Shift routine. The output is a 32-bit integer whose value depends upon the value of parm2:

- If \(\text{parm2} \geq 0\), result is a copy of parm1 shifted left by \(\text{parm2}\) bits.
- If \(\text{parm2} < 0\), result is a copy of parm1 shifted right by \(|\text{parm2}|\) bits.

In either case, vacated bits are set to 0.

CEESITST—Bit test

CEESITST selectively tests a bit in its parm1 input to determine if the bit is on.

Syntax

```plaintext
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 input is a 32-bit integer in the range \(0 \leq \text{parm2} \leq 31\).

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

The following symbolic conditions can result from this service:
result (output)
The result of the Bit Test routine. The output is a 32-bit integer with value:
1, if bit number \( parm2 \) in \( parm1 \) is 1
0, if bit number \( parm2 \) in \( parm1 \) is 0

Bits are counted from the right.
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.

This chapter contains the following sections:
- “Call interface to math services”
- “Sample calls to math services” on page 474
- “Examples of math services” on page 510
- “Math services” on page 475

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
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.

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.

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.

Sample calls to math services

This section contains sample calls to Language Environment math services from C/C++, COBOL, and PL/I.
C/C++ call to CEESLOG—logarithm base e

```c
#include <leawi.h>
#include <string.h>
#include <stdio.h>

int main (void) {
  float int1, intr;
  _FEEDBACK fc;
  #define SUCCESS "\0\0\0\0"
  int1 = 39;
  CEESSLOG(&int1,&fc,&intr);
  if (memcmp(&fc,SUCCESS,4) != 0) {
    printf("CEESSLOG failed with message number %d\n", fc.tok_msgno);
    exit(2999);
  }
  printf("Log base e of %f is %f\n",int1,intr);
}
```

COBOL call to CEESSLOG—logarithm base e

```cobol
77 ARG1RS COMP-1.
77 FBCODE PIC X(12).
77 RESLTRS COMP-1.

    CALL "CEESSLOG" USING ARG1RS, FBCODE, RESLTRS.
```

PL/I call to CEESSLOG—logarithm base e

```pli
DCL ARG1 RESULT REAL FLOAT DEC (6);
DCL FC CHARACTER (12);
CALL CEESSLOG (ARG1, FC, RESULT)
```

Math services

This section contains an alphabetical list of the Language Environment math services.

CEESxABS—Absolute value

CEESxABS returns the absolute value of the parameter by using the equation \( \text{result} = |\text{parm1}| \).

The following routines are provided for the various data types supported:

- **CEESIABS** 32-bit binary integer
- **CEESSABS** 32-bit single floating-point number
- **CEESDABS** 64-bit double floating-point number
- **CEESQABS** 128-bit extended floating-point number
- **CEESTABS** 32-bit single floating-point complex number
CEESABS

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESABS</td>
<td>64-bit double floating-point complex number</td>
</tr>
<tr>
<td>CEESRABS</td>
<td>128-bit extended floating-point complex number</td>
</tr>
</tbody>
</table>

**Syntax**

```
CEESABS(—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 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, \[ \Omega = 16^{63} (1 - 16^{-15}) \]
For extended-precision routines, \[ \Omega = 16^{63} (1 - 16^{-28}) \]

**CEESxACS—Arccosine**

CEESxACS returns the arccosine of the parameter by using the equation \[ \text{result} = \arccos(parm1). \]

The following routines are provided for the various data types supported:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESSACS</td>
<td>32-bit single floating-point number</td>
</tr>
<tr>
<td>CEESDACS</td>
<td>64-bit double floating-point number</td>
</tr>
<tr>
<td>CEESQACS</td>
<td>128-bit extended floating-point number</td>
</tr>
</tbody>
</table>

**Syntax**

```
CEESACS(—parm1—,—fc—,—result—)
```

**parm1 (input)**
The input to the arccosine routine. The input range is
|parm1| ≤ 1

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

The following symbolic conditions can result from this service:

<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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>routine-name.</td>
</tr>
</tbody>
</table>

result (output)
The result, in radians, of the arccosine routine.

0 ≤ result ≤ π

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:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESSASN</td>
<td>32-bit single floating-point number</td>
</tr>
<tr>
<td>CEESDASN</td>
<td>64-bit double floating-point number</td>
</tr>
<tr>
<td>CEESQASN</td>
<td>128-bit extended floating-point number</td>
</tr>
</tbody>
</table>

Syntax

```plaintext
>>>CEESxASN—(<parm1>,—fc—,—result—)
```

parm1 (input)
The input to the arcsine routine. The input range is

|parm1| ≤ 1

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

The following symbolic conditions can result from this service:

<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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>routine-name.</td>
</tr>
</tbody>
</table>
result (output)
The result, in radians, of the arcsine routine. The output range is

\[ |\text{result}| < \frac{\pi}{2} \]

CEESxATH—Hyperbolic arctangent
CEESxATH returns the hyperbolic arctangent of the parameter by using the equation

\[ \text{result} = \tanh^{-1}(\text{parm}1) \]

The following routines are provided for the various data types supported:
- CEESATH 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

```c
CEESxATH((parm1, fc, result))
```

parm1 (input)
The input to the hyperbolic arctangent routine.
The input range for real variables is:

\[ |\text{parm}1| \leq 1 \]

For complex variables, \(\text{parm}1\) cannot be equal to 1 or −1.

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

The following symbolic conditions can result from this service:

<table>
<thead>
<tr>
<th>Code (fc)</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 \textit{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 \textit{routine-name}.</td>
</tr>
</tbody>
</table>

result (output)
The result of the hyperbolic arctangent routine. The output range for functions of real variables is
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, \( \Omega = 16^{63}(1 - 16^{-28}) \)

**CEESxATH**—Arctangent

CEESxATH 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:

- **CEESSATN** 32-bit single floating-point number
- **CEESDATN** 64-bit double floating-point number
- **CEESQATN** 128-bit extended floating-point number
- **CEESTATN** 32-bit single floating-point complex number
- **CEESEATN** 64-bit double floating-point complex number
- **CEESRATN** 128-bit extended floating-point complex number

**Syntax**

```
>>> CEESxATH( parm1, fc, result )
```

**parm1** *(input)*

The input to the arctangent routine. The input range for real variables is not restricted. The input range of complex variables in parm1 is not equal to i or -i, where

\[
i = \sqrt{-1}
\]

**fc** *(output)*

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

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:

\[
|\text{result}| \leq \Omega
\]
**CEESxAT2—Arctangent2**

CEESxAT2 calculates a result by using the equation \( \text{result} = \text{the angle (in radians)} \) between the positive X axis and a vector defined by \((\text{parm2}, \text{parm1})\) with a range from

\(-\pi\) to \(\pi\), with \(\pi\),

For example, if \(\text{parm1}\) and \(\text{parm2}\) are positive, then \(\text{result} = \arctan(\text{parm1}/\text{parm2})\).

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**

```plaintext
CEESxAT2(parm1, parm2, fc, result)
```

**parm1 (input)**

The first input to the arctangent2 routine. The input range is \(\text{parm1}\) cannot equal 0 if \(\text{parm2}\) equals 0.

**parm2 (input)**

The second parameter to the arctangent2 routine. The input range is \(\text{parm2}\) cannot equal 0 if \(\text{parm1}\) 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 (\text{limit}) in math routine (\text{routine-name}).</td>
</tr>
<tr>
<td>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine (\text{routine-name}).</td>
</tr>
</tbody>
</table>

**result (output)**

The result, in radians, of the arctangent2 routine. The output range is

\(|\text{result}| \leq \pi/2\)

**CEESxCJG—Conjugate of complex**

CEESxCJG returns the conjugate of the complex number by using the equation \(\text{result} = u - vi\), where \(\text{parm1} = u + vi\).

The following routines are provided for the various data types supported:
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>

The result of the conjugate of complex routine.

**CEESxCOS—Cosine**

CEESxCOS returns the cosine of the parameter by using the equation \( \text{result} = \cos(parm1) \).

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

The input range for real variables is:

- for single floating-point numbers,
  
  \( |\text{parm1}| < (2^{18} \cdot \Pi) \)

- for double floating-point numbers, and
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>
<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>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

\[ |result| \leq 1 \]

**CEESxCSH—Hyperbolic cosine**
CEESxCSH returns the hyperbolic cosine of the parameter by using the equation

\[ result = \cosh(parm1) \]

The following routines are provided for the various data types supported:
- **CEESSCSH** 32-bit single floating-point number
- **CEESDCSH** 64-bit double floating-point number
- **CEESQCSH** 128-bit extended floating-point number
- **CEESTCSH** 32-bit single floating-point complex number
- **CEESECSH** 64-bit double floating-point complex number
- **CEESRCSSH** 128-bit extended floating-point complex number

**Syntax**

```
>>>CEESxCSH—(parm1,—fc—,—result—)
```

**parm1 (input)**
The input to the hyperbolic cosine routine.
The input range for the functions of real variables is:

\[ |parm1| < 175.366 \]

For complex functions, the input range differs for the imaginary and real parts. For the real part of complex numbers, it is

\[ |\text{Re (parm1)}| < 174.673 \]

For the imaginary part of complex numbers, the input range is
A 12-byte feedback code, 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>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 cosine routine. The output range for functions of real variables is

\[ 1 \leq \text{result} \leq \Omega \]

for single-precision routines,

\[ \Omega = 16^{63}(1 - 16^6) \]

\[ 16^{63}(1 - 16^{-14}) \]

for double-precision routines, and for extended-precision routines.

\[ 16^{63}(1 - 16^{-28}) \]

CEESxCTN—Cotangent

CEESxCTN returns the cotangent of the parameter by using the equation \( \text{result} = \cot(parm1) \).
The following routines are provided for the various data types supported:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESSCTN</td>
<td>32-bit single floating-point number</td>
</tr>
<tr>
<td>CEESDCTN</td>
<td>64-bit double floating-point number</td>
</tr>
<tr>
<td>CEESQCTN</td>
<td>128-bit extended floating-point number</td>
</tr>
</tbody>
</table>

**Syntax**

\[
\text{CEESxCTN}(\text{parm1}, \text{fc}, \text{result})
\]

**parm1 (input)**

The input, in radians, into the cotangent routine.

If \( \text{parm1} \) is a 32-bit single floating-point number, input range is

\[
|\text{parm1}| < \left(2^{18} \cdot \pi\right)
\]

If \( \text{parm1} \) is a 64-bit double floating-point number, input range is

\[
|\text{parm1}| < \left(2^{50} \cdot \pi\right)
\]

If \( \text{parm1} \) is a 128-bit extended floating-point number, the input range is

\[
|\text{parm1}| < 2^{100}
\]

If this is an extended floating-point number, this argument cannot approach a multiple of \( \pi \). Single floating-point numbers and double floating-point numbers cannot approach zero.

**fc (output)**

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

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 ( \text{routine-name} ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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>
</tbody>
</table>

**result (output)**

The result of the cotangent routine. The output range is

\[
|\text{result}| \leq \Omega
\]
The output range is
\[
result \leq \Omega
\]

**CEESxDIM—Positive difference**

CEESxDIM returns the positive difference between two numbers by using one of the following equations:

if
then \( result = parm1 - parm2 \)

\( parm1 > parm2 \)

if
then \( result = 0 \)

\( parm1 \leq parm2 \)

The following routines are provided for the various data types supported:

- **CEESIDIM** 32-bit binary integer
- **CEESSDIM** 32-bit single floating-point number
- **CEESDDIM** 64-bit double floating-point number
- **CEESQDIM** 128-bit extended floating-point number

**Syntax**

```
CEESxDIM parm1 parm2 fc result
```

\( parm1 \) (input)
The first input to the positive difference routine. The input range is not restricted.

\( parm2 \) (input)
The second parameter to the positive difference routine. The input range is not restricted.

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

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 \[ \leq \Omega \]

For single-precision routines, \( \Omega = 16^{63}(1 - 16^{-6}) \)
For double-precision routines, \( \Omega = 16^{63}(1 - 16^{-14}) \)
For extended-precision routines, \( \Omega = 16^{63}(1 - 16^{-28}) \)

**CEESxDVD—Floating-point complex divide**
CEESxDVD performs the mathematical function of floating-point complex divide by using the equation:

\[
\text{result} = \frac{\text{parm1}}{\text{parm2}}
\]

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

**Syntax**

```
CEESxDVD(parm1, parm2, fc, result)
```

**parm1 (input)**
The first input to the math service. Any representable complex number can be used as input.

**parm2 (input)**
The second parameter to the math service. Do not set \( \text{parm2} \) 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>0</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:
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

```
CEEESxERC(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:

\[
result = \left(1 - \frac{2}{\pi} \int_{0}^{\infty} \frac{e^{-u^2}}{u} \, du\right)
\]

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

```
CEEESxERF(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

$|result| \leq 1$

**CEESxEXP—Exponential base e**

CEESxEXP calculates the mathematical function of $e$ raised to a power by using the equation:

$$result = e^{parm1}$$

The following routines are provided for the various data types supported:

- **CEESSEXP** 32-bit single floating-point number
- **CEESDEXP** 64-bit double floating-point number
- **CEESQEXP** 128-bit extended floating-point number
- **CEESTEXP** 32-bit single floating-point complex number
- **CEESEEXP** 64-bit double floating-point complex number
- **CEESREXP** 128-bit extended floating-point complex number

**Syntax**

```plaintext
|CEESxEXP| (parm1, fc, result)
```

**parm1 (input)**
The input to the exponential base e routine. The input range for functions of real variables is:

$|parm1| \leq 174.673$

For complex functions, the input range differs for the imaginary and real parts of the function. The input ranges are for the real part of complex numbers:

$|\text{Re } (parm1)| < 174.673$

For the imaginary part, the ranges are:
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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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

\[ 0 < \text{result} \leq \Omega \]

For single-precision routines, \( \Omega = 16^{63}(1 - 16^{-6}) \)

For double-precision routines, \( \Omega = 16^{63}(1 - 16^{-14}) \)

For extended-precision routines, \( \Omega = 16^{63}(1 - 16^{-28}) \)

**CEESxGMA—Gamma function**

CEESxGMA performs the mathematical gamma function by using the equation:
The following routines are provided for the various data types supported:

- **CEESSGMA** 32-bit single floating-point number
- **CEESDGMA** 64-bit double floating-point number

### Syntax

```c
result = \int_{0}^{\infty} u_{parm1}^{-1} e^{-u} du
```

The following routines are provided for the various data types supported:

```c
para1 (input)
```

The input to the gamma function. The input range is

\[2^{252} < \text{parm1} < 57.5744\]

```c
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 (\text{range}) in math routine (\text{routine-name}).</td>
</tr>
</tbody>
</table>

```c
result (output)
```

The result of the gamma function. The output range is

\[0.88560 \leq \text{result} \leq \Omega\]

\[\Omega = 16^{63} (1-16^{-6})\]

for single-precision routines and

\[\Omega = 16^{63} (1-16^{-14})\]

for double-precision routines.

### CEESxIMG—Imaginary part of complex

CEEESxIMG returns the imaginary part of a complex number using the equation

\[\text{result} = \nu, \text{ where } \text{parm1} = u + \nu i.\]

The following routines are provided for the various data types supported:
CEESxIMG

CEESTIMG  32-bit single floating-point complex number
CEESEIMG  64-bit double floating-point complex number
CEESSRIMG 128-bit extended floating-point complex number

Syntax

```
$\text{C}_{}\text{EESxIMG}(\text{parm1}, \text{fc}, \text{result})$
```

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:

$$\text{result} = (\text{sign of parm1}) \cdot n, \text{ where } n = |\text{parm1}|$$

$|\text{parm1}| = |m|$

where $M$ is the greatest integer satisfying the relationship and the result is expressed as a floating-point number.

$|m| \leq |\text{parm1}|$

The following routines are provided for the various data types supported:

- CEESSSINT  32-bit single floating-point number
- CEESDINT   64-bit double floating-point number
- CEESQINT   128-bit extended floating-point number
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| ≤ Ω

For single-precision routines, Ω = 16^{63}(1 - 16^{-6})
For double-precision routines, Ω = 16^{63}(1 - 16^{-8})
For extended-precision routines, Ω = 16^{63}(1 - 16^{-28})

**CEESxLGM—Log gamma**

CEESxLGM performs the mathematical function of log gamma by using the equation:

\[
\text{result} = \log_e \Gamma(\text{parm1})
\]

or

\[
\text{result} = \log_e \int_0^{\infty} u^{\text{parm1} - 1} e^{-u} \, du
\]

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**

```plaintext
>>>CEESxLGM—(parm1--, fc--, result—)
```
**CEESxLGM**

*parm1* (input)

The input to the log gamma routine. The input range is

\[ parm1 < 4.2913 \times 10^73 \]

*fc* (output)

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

The following symbolic conditions can result from this service:

<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 log gamma routine. The output range is

\[ -0.12149 < result < \Omega \]

for single-precision routines and

\[ \Omega = 16^{63} (1 - 16^{-6}) \]

\[ \Omega = 16^{63} (1 - 16^{-14}) \]

for double-precision routines.

**CEESxLG1—Logarithm base 10**

CEESxLG1 returns the logarithm base 10 of the input parameter by using the equation

\[ result = \log_{10} parm1 \]

The following routines are provided for the various data types supported:

- **CEESSLG1**: 32-bit single floating-point number
- **CEESDLG1**: 64-bit double floating-point number
- **CEESQLG1**: 128-bit extended floating-point number

**Syntax**

```
>>>CEESxLG1((parm1, fc, result))
```

*parm1* (input)

The input to the log base 10 routine. The input range is *parm1 > 0*. 

---

494  z/OS V1R6.0 Language Environment Programming Reference
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

\[-78.268 \leq \text{result} \leq 75.859\]

**CEESxLG2—Logarithm base 2**

CEESxLG2 performs the mathematical function logarithm base 2 by using the equation

\[
\text{result} = \log_2 \text{parm1}
\]

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**

```
CEE5xLG2(parm1, fc, result)
```

**parm1 (input)**

The input to the log base 2 routine. The input range is \( \text{parm1} > 0 \).

**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 2 routine. The output range is

\[-260 \leq \text{result} \leq 252\]
CEESxLOG

CEESxLOG—Logarithm base e

CEESxLOG performs the mathematical function logarithm base e by using the equation

\[ \text{result} = \log_e \text{parm1} \quad (\text{result} = \ln \text{parm1}) \]

The following routines are provided for the various data types supported:
- CEESLOG: 32-bit single floating-point number
- CEESSLOG: 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**

```c
CEESxLOG(parm1, fc, result)
```

**parm1 (input)**

The input to the log base e routine. The input range for reals is \( \text{parm1} > 0 \).

**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>
<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

\[-180.218 \leq \text{result} \leq 174.673\]
CEESxMLT—Floating-point complex multiply

CEESxMLT performs the mathematical function floating-point complex multiply by using the equation

\[ \text{result} = \text{parm1} \times \text{parm2} \]

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**

```
CEESxMLT(par1, par2, fc, result)
```

- **par1** (*input*)
  The first input to the math service. Any representable complex number can be used as input.

- **par2** (*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
The following routines are provided for the various data types supported:

- **CEESIMOD** 32-bit binary integer
- **CEESSMOD** 32-bit single floating-point number
- **CEESDMOD** 64-bit double floating-point number
- **CEESQMOD** 128-bit extended floating-point number

### Syntax

```
CEESxMOD(parm1, parm2, fc, result)
```

**parm1 (input)**

The first parameter to the modular arithmetic routine. The input range is not restricted.

**parm2 (input)**

The second parameter to the modular arithmetic routine. The input range is

\[ parm2 \neq 0 \]

If \( parm2 = 0 \), the modulus routine is undefined. In addition, a divide exception is recognized and an interrupt occurs.

**fc (output)**

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

The following symbolic conditions can result from this service:

<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

\[ |result| \leq \Omega \]
CEESxNIN—Nearest integer

CEESxNIN performs the mathematical function nearest integer by using the equation:

\[ \text{result} = (\text{sign of } \text{parm1}) \cdot n \]

If \( \text{parm1} \geq 0 \), then \( n = \lfloor \text{parm1} + 0.5 \rfloor \)

If \( \text{parm1} < 0 \), then \( n = \lceil \text{parm1} - 0.5 \rceil \)

\[ n = |m|, \text{ where } m \text{ is the greatest integer satisfying the relationship } |m| \leq |\text{ parm1 } + 0.5|, \]

or \( |m| \leq |\text{ parm1 } - 0.5|, \) respectively.

The following routines are provided for the various data types supported:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEESSNIN</td>
<td>32-bit single floating-point number</td>
</tr>
<tr>
<td>CEESDNIN</td>
<td>64-bit double floating-point number</td>
</tr>
</tbody>
</table>

Syntax

```
>CESxxNIN((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

\[ \text{result} = (\text{sign of } \text{parm1}) \cdot v \]
If $\text{parm1} \geq 0$, then $v = \lfloor \text{parm1} + .5 \rfloor$. If \( \text{parm1} < 0 \), then $v = \lfloor \text{parm1} - .5 \rfloor$.

$v = |m|$, where $m$ is the greatest integer satisfying the relationship $|m| \leq |\text{parm1} + .5|$, or $|m| \leq |\text{parm1} - .5|$, respectively.

and the resulting $v$ is expressed as a floating-point number.

The following routines are provided for the various data types supported:

- **CEESSNWN** 32-bit single floating-point number
- **CEESDNWN** 64-bit double floating-point number

### Syntax

```plaintext
CEESSNWN(parml, 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 is $|\text{result}| \leq \Omega$

$$
\Omega = 16^{63} \left(1 - 16^{-6}\right) 
$$

for single-precision routines, and

$$
\Omega = 16^{63} \left(1 - 16^{-14}\right) 
$$

for double-precision routines.

### CEESxSGN—Transfer of sign

CEESxSGN performs the mathematical function transfer of sign by using either of the two equations:
result = |parm1| if parm2 ≥ 0 or result = -|parm1| if parm2 < 0.

The following routines are provided for the various data types supported:
- **CEESISGN** 32-bit binary integer
- **CEESSSGN** 32-bit single floating-point number
- **CEESDSGN** 64-bit double floating-point number
- **CEESQSGN** 128-bit extended floating-point number

**Syntax**

```plaintext
CEEESxSGN(parm1, parm2, fc, result)
```

`parm1 (input)`
The first input to the transfer of sign routine. The input range is not restricted.

`parm2 (input)`
The second parameter to the transfer of sign routine. The input range is not restricted.

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

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

|result| ≤ Ω

For single-precision routines, Ω = 16^{63}(1 - 16^{-6})
For double-precision routines, Ω = 16^{63}(1 - 16^{-10})
For extended-precision routines, Ω = 16^{63}(1 - 16^{-20})

**CEESxSIN—Sine**

CEESxSIN returns the sine of the parameter by using the equation result = sin (parm1).

The following routines are provided for the various data types supported:
- **CEESSSIN** 32-bit single floating-point number
- **CEESDSIN** 64-bit double floating-point number
- **CEESQSin** 128-bit extended floating-point number
- **CEESTSIN** 32-bit single floating-point complex number
- **CEESESIN** 64-bit double floating-point complex number
- **CEESRSIN** 128-bit extended floating-point complex number
**Syntax**

```
CEESxSIN( parm1, fc, result )
```

*parm1 (input)*

The input, in radians, to the sine routine.

For real functions, if *parm1* is a 32-bit single floating-point number, the input range is

\[ |\text{parm1}| < (2^{18} \cdot \pi) \]

If *parm1* is a 64-bit double floating-point number, the input range is

\[ |\text{parm1}| < (2^{50} \cdot \pi) \]

If *parm1* is an extended floating-point number, the input range is

\[ |\text{parm1}| < 2^{100} \]

For complex functions, the input range differs for the imaginary and real parts of the input. For the imaginary part, the input range is

\[ |\text{Im}(\text{parm1})| < 174.673 \]

For the real part, it is

\[ |\text{Re}(\text{parm1})| < (2^{18} \cdot \pi) \]

for single floating-point complex numbers,

\[ |\text{Re}(\text{parm1})| < (2^{50} \cdot \pi) \]

for double floating-point complex numbers, and

\[ |\text{Re}(\text{parm1})| < 2^{100} \]

for extended floating-point complex numbers.

*fc (output)*

A 12-byte feedback code, optional in some languages, that indicates the result of this service.
The following symbolic conditions can result from this service:

<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>
<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 sine routine. The output range for functions of real variables is

\[-1 \leq \text{result} \leq 1\]

**CEESxSNH—Hyperbolic sine**

CEESxSNH performs the mathematical function hyperbolic sine by using the equation \(\text{result} = \sinh(parm1)\).

The following routines are provided for the various data types supported:

- **CEESSSNH**: 32-bit single floating-point number
- **CEESDSNH**: 64-bit double floating-point number
- **CEESQSNH**: 128-bit extended floating-point number
- **CEESTSNH**: 32-bit single floating-point complex number
- **CEESESNH**: 64-bit double floating-point complex number
- **CEESRSNH**: 128-bit extended floating-point complex number

**Syntax**

```plaintext
>>>CEESxSNH—(parm1—,—fc—,—result—)
```

**parm1 (input)**

The input to the hyperbolic sine routine. Input range for reals is:

\[|\text{parm1}| < 175.366\]

For complex functions, the input range differs for the imaginary and real parts of the input. For the real part, the input range is:

\[|\text{Re(parm1)}| < 174.673\]

The input range of the imaginary part differs depending on the precision of \(\text{parm1}\):
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 of the hyperbolic sine routine. The output range for functions of real variables is

\[ |\text{Im}(\text{parm1})| \leq 2 \pi \]

CEESxSQT—Square root

CEESxSQT returns the square root of \( \text{parm1} \) 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
- CEESRSQT 128-bit extended floating-point complex number
**Syntax**

\[ \text{CEEESxSQT} \left( \text{parm1}, \text{fc}, \text{result} \right) \]

**parm1 (input)**

The input to the square root routine. The input range for real number functions is

\[ \text{parm1} \geq 0 \]

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

\[ 0 \leq \text{result} \leq \Omega^{1/2} \]

For single-precision routines, \( \Omega = 16^{63}(1 -16^{-6}) \)
For double-precision routines, \( \Omega = 16^{63}(1 -16^{-14}) \)
For extended-precision routines, \( \Omega = 16^{63}(1 -16^{-30}) \)

**CEESxTAN—Tangent**

CEESxTAN returns the tangent of the parameter by using the equation \( \text{result} = \tan(\text{parm1}) \).

The following routines are provided for the various data types supported:

- **CEESSTAN** 32-bit single floating-point number
- **CEESDTAN** 64-bit double floating-point number
- **CEESQTAN** 128-bit extended floating-point number
- **CEESSTTAN** 32-bit single floating-point complex number
- **CEESETAN** 64-bit double floating-point complex number
- **CEESRTAN** 128-bit extended floating-point complex number
CEESxTAN

**Syntax**

\[
\text{CEESxTAN}(\text{parm1}, \text{fc}, \text{result})
\]

**parm1 (input)**

The input, in radians, to the tangent routine.

If \(\text{parm1}\) is a single floating-point number, the input range is:

\[|\text{parm1}| < 2^{18} \cdot \pi\]

If \(\text{parm1}\) is a double floating-point number, the input range is:

\[|\text{parm1}| < 2^{50} \cdot \pi\]

If \(\text{parm1}\) is an extended floating-point number, the input range is:

\[|\text{parm1}| < 2^{100}\]

Also, for extended as well 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 (\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>CEE1V9</td>
<td>1</td>
<td>2025</td>
<td>An underflow has occurred in math routine (\text{routine-name}).</td>
</tr>
</tbody>
</table>

**result (output)**

The result of the tangent routine. The output range for functions of real variables is
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
- **CEESTTNH**: 32-bit single floating-point complex number
- **CEESETNH**: 64-bit double floating-point complex number
- **CEESRTNH**: 128-bit extended floating-point complex number

Syntax

\[
\text{CEESxTNH} (\text{parm1}, -\text{fc} -, \text{result})
\]

\( \text{parm1 (input)} \)

The input to the hyperbolic tangent routine. The input range is not restricted for real functions. For complex functions, \( \text{parm1} \) must not approach odd multiples of \( \Pi/2 \times i \), where \( i = \sqrt{-1} \)

\[
i = \sqrt{-1}
\]

\( \text{fc (output)} \)

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>

\( \text{result (output)} \)

The result of the hyperbolic tangent routine. The output range for functions of real variables is:

\[ |\text{result}| \leq \Omega \]

CEESxXPx—Exponentiation

CEESxXPx performs the mathematical function exponentiation by using the equation:

\[
|\text{result}| 
\]

\[ \leq \Omega \]
The following routines are provided for the various data types supported:

- **CEESIXPI**: 32-bit binary integer raised to a 32-bit binary integer
- **CEESSXPI**: 32-bit single floating-point number raised to a 32-bit binary integer
- **CEESDXPI**: 64-bit double floating-point number raised to a 32-bit binary integer
- **CEESQXPI**: 128-bit extended floating-point number raised to a 32-bit binary integer
- **CEESTXPI**: 32-bit single floating-point complex number raised to a 32-bit binary integer
- **CEESEXPI**: 64-bit double floating-point complex number raised to a 32-bit binary integer
- **CEESRXPI**: 128-bit extended floating-point complex number raised to a 32-bit binary integer
- **CEESSXPS**: 32-bit single floating-pointing point raised to a 32-bit single floating-point
- **CEESDXPD**: 64-bit double floating-point raised to a 64-bit double floating-point
- **CEESQXPQ**: 128-bit extended floating-point raised to a 128-bit extended floating-point
- **CEESTXPT**: 32-bit single floating-point complex raised to a 32-bit single floating-point complex
- **CEESEXPE**: 64-bit double floating-point complex raised to a 64-bit double floating-point complex
- **CEESRXPR**: 128-bit extended floating-point complex raised to a 128-bit extended floating-point complex

### Syntax

```plaintext
CEESxXPx( parm1, parm2, fc, result )
```

**parm1** *(input)*
The input for the base of the exponentiation routine. The input range for functions of real variables is as follows:

*if* *parm1* = 0, then *parm2* > 0.

If *parm1* is a 32-bit number and *parm1* < 0, then
and *parm2* = 'awholenumber'.

\[ |parm1|/(16^6 - 1) \]

If *parm1* is a 64-bit number and *parm1* < 0, then
and *parm2* = 'awholenumber'.

\[ |Parm1|/(16^{14} - 1) \]

If *parm1* is a 128-bit number and *parm1* < 0, then
and *parm2* = 'awholenumber'.

\[ |Parm1|/(16^{28} - 1) \]

The input range for functions of complex variables: If \( \text{Re}(parm1) = 0 \) and \( \text{Im}(parm1) = 0 \), then \( \text{Re}(parm2) \) 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>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 routine-name.</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 routine-name.</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>
<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 routine-name.</td>
</tr>
<tr>
<td>CEE1UN</td>
<td>2</td>
<td>2007</td>
<td>The exponent exceeded limit in math routine routine-name.</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 routine-name.</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 limit in math routine routine-name.</td>
</tr>
<tr>
<td>CEE1V5</td>
<td>2</td>
<td>2021</td>
<td>For an exponentiation operation (X**Y), the parameter combination of Y*log2(X) generated a number greater than or equal to limit in math routine routine-name.</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 routine-name. 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
Examples of math services

This section contains representative calls to math services from each supported language.

C/C++ math service example

The following are C/C++ examples.

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

The following are COBOL examples.
Log base $e$ (CEESSLOG)

77 ARG1RS COMP-1.
77 FBCODE PIC X(12).
77 RESLTRS COMP-1.

CALL "CEESSLOG" USING ARG1RS, FBCODE, RESLTRS.

Log base 10 (CEESDLG1)

77 ARG1RL COMP-2.
77 FBCODE PIC X(12).
77 RESLTRL COMP-2.

CALL "CEESDLG1" USING ARG1RL, FBCODE, RESLTRL.

Exponentiation (CEESIXPI)

77 ARG1IS PIC S9(9) COMP.
77 ARG2IS PIC S9(9) COMP.
77 FBCODE PIC X(12).
77 RESLTIS PIC S9(9) COMP.

CALL "CEESIXPI" USING ARG1IS, ARG2IS, FBCODE, RESLTIS.

Exponentiation (CEESSXPI)

77 ARG1RS COMP-1.
77 ARG2IS PIC S9(9) COMP.
77 FBCODE PIC X(12).
77 RESLTRS COMP-1.

CALL "CEESSXPI" USING ARG1RS, ARG2IS, FBCODE, RESLTRS.

Arctangent2 (CEESSAT2)

77 ARG1RS COMP-1.
77 ARG2RS COMP-1.
77 FBCODE PIC X(12).
77 RESLTRS COMP-1.

CALL "CEESSAT2" USING ARG1RS, ARG2RS, FBCODE, RESLTRS.

PL/I math service examples

The following are PL/I examples.
Modular arithmetic and log base e (CEESIMOD and CEESSLOG)

PLIMATH: PROC OPTIONS(MAIN);
   DCL CEESSLOG ENTRY OPTIONS(ASM) EXTERNAL;
   DCL CEESIMOD ENTRY OPTIONS(ASM) EXTERNAL;

   DCL ARG1 RESULT REAL FLOAT DEC (6);
   DCL ARGM1 ARGM2 RES2 FLOAT BINARY(21)
   DCL FC CHARACTER (12);

   /* Call log base e routine, which has */
   /* only one input parameter */
   CALL CEESSLOG (ARG1, FC, RESULT)
     IF ( FC = '000000000000000000000000'X )
       THEN DO;
       PUT SKIP LIST
        ( 'Error occurred in call to CEESSLOG.' );
       ELSE;
     END;

   /* Call modular arithmetic routine, */
   /* which has two input parameters */
   CALL CEESIMOD (ARGM1, ARGM2, FC, RES2);

   IF ( FC = '000000000000000000000000'X )
     THEN DO;
     PUT SKIP LIST
      ( 'Error occurred in call to CEESIMOD.' );
     ELSE;
     END;

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;

Double-precision complex tangent (CEESETAN)
Part 3. Appendixes
Appendix A. IBM-supplied country code defaults

Table 28 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 101 for more information.

**Note:** In the table, some currency symbols are shown as X'9F404040'. An @euro codeset modifier can be used to get (euro-sign) instead of currency. How this is displayed differs; in the United States, it is shown as a '$' 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>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>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>X'9F404040'</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>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>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>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>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Argentina</td>
<td>AR</td>
<td>,</td>
<td>,</td>
<td>A</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Australia</td>
<td>AU</td>
<td>,</td>
<td>,</td>
<td>$</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>S</td>
<td>HH:MI:SS,999</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Bahamas</td>
<td>BS</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</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>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>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Barbados</td>
<td>BB</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Belgium</td>
<td>BE</td>
<td>,</td>
<td>,</td>
<td>BF</td>
<td>HH:MI:SS,999</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Benin</td>
<td>BJ</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</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>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>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>,</td>
<td>Din</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Botswana</td>
<td>BW</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
<td>,</td>
<td>,</td>
<td>NCzS</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Brunei</td>
<td>BN</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</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>,</td>
<td>Lv</td>
<td>HH:MI:SS</td>
<td>YYYY-RRRZ-DD</td>
<td>YYYY-RRRZ-DD HH:MI:SS</td>
</tr>
<tr>
<td>Burkina Faso (Upper Volta)</td>
<td>BF</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Burma</td>
<td>BU</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Canada</td>
<td>CA</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>HH:MI:SS,99</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Cayman Islands</td>
<td>KY</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</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>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Chile</td>
<td>CL</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Colombia</td>
<td>CO</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MI:SS AP</td>
</tr>
<tr>
<td>Croatia</td>
<td>HR</td>
<td>,</td>
<td>,</td>
<td>Din</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Cuba</td>
<td>CU</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Cyprus</td>
<td>CY</td>
<td>,</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Czech Republic</td>
<td>CZ</td>
<td>,</td>
<td>,</td>
<td>XD247A240'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI: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>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>
</tr>
<tr>
<td>Dominican Republic</td>
<td>DO</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS AP</td>
</tr>
<tr>
<td>Ecuador</td>
<td>EC</td>
<td>,</td>
<td>,</td>
<td>$</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Egypt</td>
<td>EG</td>
<td>,</td>
<td>,</td>
<td>EG</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>El Salvador</td>
<td>SV</td>
<td>,</td>
<td>,</td>
<td>c/</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Estonia</td>
<td>EE</td>
<td>,</td>
<td>,</td>
<td>Kr</td>
<td>HH:MI:SS</td>
<td>DD-MM-YYYY</td>
<td>DD-MM-YYYY HH:MI:SS</td>
</tr>
<tr>
<td>Ethiopia</td>
<td>ET</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Finland</td>
<td>FI</td>
<td>,</td>
<td>,</td>
<td>Mk</td>
<td>HH:MI:SS.999</td>
<td>DD.MM/YYYY</td>
<td>DD.MM/YYYY HH:MI:SS.999</td>
</tr>
<tr>
<td>France</td>
<td>FR</td>
<td>,</td>
<td>,</td>
<td>F</td>
<td>HH:MI:SS.9</td>
<td>DD.MM.YYYY</td>
<td>DD.MM.YYYY HH:MI:SS.9</td>
</tr>
<tr>
<td>Germany</td>
<td>DE</td>
<td>,</td>
<td>,</td>
<td>DM</td>
<td>HH:MI:SS</td>
<td>DD.MM.YYYY</td>
<td>DD.MM.YYYY HH:MI:SS</td>
</tr>
<tr>
<td>Greece</td>
<td>GR</td>
<td>,</td>
<td>,</td>
<td>Drs</td>
<td>HH:MI:SS.999</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS.999</td>
</tr>
<tr>
<td>Guatemala</td>
<td>GT</td>
<td>,</td>
<td>,</td>
<td>Q</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Guinea- Bissau</td>
<td>GW</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Guyana</td>
<td>GY</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Haiti</td>
<td>HT</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Honduras</td>
<td>HN</td>
<td>,</td>
<td>,</td>
<td>L</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS AP</td>
</tr>
<tr>
<td>China (Hong Kong S.A.R.)</td>
<td>HK</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Hungary</td>
<td>HU</td>
<td>,</td>
<td>,</td>
<td>FT</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Iceland</td>
<td>IS</td>
<td>,</td>
<td>,</td>
<td>kr</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>India</td>
<td>IN</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Indonesia</td>
<td>ID</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Iran</td>
<td>IR</td>
<td>,</td>
<td>,</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Iraq</td>
<td>IQ</td>
<td>,</td>
<td>,</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Ireland</td>
<td>IE</td>
<td>,</td>
<td>,</td>
<td>X’5B404040'</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Israel</td>
<td>IL</td>
<td>,</td>
<td>,</td>
<td>NIS</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Italy</td>
<td>IT</td>
<td>,</td>
<td>,</td>
<td>L</td>
<td>HH:MI:SS.999</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS.999</td>
</tr>
<tr>
<td>Jamaica</td>
<td>JM</td>
<td>,</td>
<td>,</td>
<td>X’9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Japan</td>
<td>JP</td>
<td>,</td>
<td>,</td>
<td>X’5B404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Jordan</td>
<td>JO</td>
<td>,</td>
<td>,</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Kuwait</td>
<td>KW</td>
<td>,</td>
<td>,</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Latvia</td>
<td>LV</td>
<td>,</td>
<td>,</td>
<td>Ls</td>
<td>HH:MI:SS</td>
<td>YYYY-DD:RRRZ</td>
<td>YYYY-DD:RRRZ HH:MI:SS</td>
</tr>
<tr>
<td>Lebanon</td>
<td>LB</td>
<td>,</td>
<td>,</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</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>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Liechtenstein</td>
<td></td>
<td></td>
<td></td>
<td></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>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>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>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>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI: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>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>
</tr>
<tr>
<td>Malawi</td>
<td>MW</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</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>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>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>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>$</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MI:SS AP</td>
</tr>
<tr>
<td>Monaco</td>
<td>MC</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Morocco</td>
<td>MA</td>
<td>.</td>
<td>.</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD DD:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Namibia</td>
<td>NA</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Netherlands Antilles</td>
<td>AN</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>New Caledonia</td>
<td>NC</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>New Zealand</td>
<td>NZ</td>
<td>.</td>
<td>.</td>
<td>$</td>
<td>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>HH:MI:SS</td>
<td>DD/MMYY</td>
<td>DD/MMYY HH:MI:SS</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
<td>.</td>
<td>.</td>
<td>kr</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>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>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Paraguay</td>
<td>PY</td>
<td>.</td>
<td>.</td>
<td>Gs</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'SB404040'</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>/</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>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'E99A4040'</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>Esc</td>
<td>HH:MI:SS</td>
<td>DD-MM-YYYY</td>
<td>DD-MM-YYYY HH:MI:SS</td>
</tr>
<tr>
<td>Puerto Rico</td>
<td>PR</td>
<td>.</td>
<td>.</td>
<td>$</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MI:SS AP</td>
</tr>
<tr>
<td>Qatar</td>
<td>QA</td>
<td>.</td>
<td>.</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/SS</td>
<td>YYYY/MM/DD DD:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Taiwan</td>
<td>TW</td>
<td>.</td>
<td>.</td>
<td>$</td>
<td>HH:MI:SS,999</td>
<td>YY/MM/DD</td>
<td>YY/MM/DD DD:MI:SS,999</td>
</tr>
<tr>
<td>Romania</td>
<td>RO</td>
<td>.</td>
<td>.</td>
<td>Lei</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Russia</td>
<td>RU</td>
<td>.</td>
<td>.</td>
<td>Rub</td>
<td>HH:MI:SS</td>
<td>DD mmm. YYYY g.</td>
<td>DD mmm. YYYY g. HH:MI:SS</td>
</tr>
<tr>
<td>Saint Lucia</td>
<td>LC</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Saudi Arabia</td>
<td>SA</td>
<td>.</td>
<td>.</td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Seychelles</td>
<td>SC</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Sierra Leone</td>
<td>SL</td>
<td>.</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Slovenia</td>
<td>SI</td>
<td>.</td>
<td>.</td>
<td>tol</td>
<td>HH:MI:SS</td>
<td>DD.MM.YYYY</td>
<td>DD.MM.YYYY HH:MI:SS</td>
</tr>
</tbody>
</table>
### Table 28. 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>Time Picture String</th>
<th>Date Picture String</th>
<th>Date and Time Picture String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudan</td>
<td>SD</td>
<td>,</td>
<td>.</td>
<td></td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>Swaziland</td>
<td>SZ</td>
<td>,</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Sweden</td>
<td>SE</td>
<td>,</td>
<td>.</td>
<td>kr</td>
<td>kl HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD kl HH:MI:SS</td>
</tr>
<tr>
<td>Switzerland</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>Syria</td>
<td>SY</td>
<td>,</td>
<td>.</td>
<td></td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Tanzania</td>
<td>TZ</td>
<td>,</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Thailand</td>
<td>TH</td>
<td>,</td>
<td>.</td>
<td>X'70404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Togo</td>
<td>TG</td>
<td>,</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Tunisia</td>
<td>TN</td>
<td>,</td>
<td>.</td>
<td></td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>Turkey</td>
<td>TR</td>
<td>,</td>
<td>.</td>
<td>TL</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Uganda</td>
<td>UG</td>
<td>,</td>
<td>.</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>Union of Soviet Socialist Republics (See note below)</td>
<td>SU</td>
<td>,</td>
<td>.</td>
<td>Rub</td>
<td>HH:MI:SS</td>
<td>DD mmm. YYYY g.</td>
<td>DD mmm. YYYY g. HH:MI:SS</td>
</tr>
<tr>
<td>United Arab Emirates</td>
<td>AE</td>
<td></td>
<td></td>
<td></td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>GB</td>
<td></td>
<td></td>
<td>X'5B404040'</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
</tr>
<tr>
<td>Uruguay</td>
<td>UY</td>
<td>,</td>
<td>N$</td>
<td>HH:MI:SS</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Vanuatu</td>
<td>VU</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Venezuela</td>
<td>VE</td>
<td>,</td>
<td>Bs.</td>
<td>ZH:MI:SS AP</td>
<td>DD/MM/YY</td>
<td>DD/MM/YY ZH:MI:SS AP</td>
<td></td>
</tr>
<tr>
<td>Western Samoa</td>
<td>WS</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Yemen</td>
<td>YE</td>
<td>,</td>
<td></td>
<td></td>
<td>HH:MI:SS</td>
<td>YYYY/MM/DD</td>
<td>YYYY/MM/DD HH:MI:SS</td>
</tr>
<tr>
<td>Yugoslavia</td>
<td>YU</td>
<td>,</td>
<td>Din</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Zaire</td>
<td>ZR</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Zambia</td>
<td>ZM</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
<td></td>
</tr>
<tr>
<td>Zimbabwe</td>
<td>ZW</td>
<td>,</td>
<td>X'9F404040'</td>
<td>HH:MI:SS</td>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD HH:MI:SS</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

The Czechoslovakia country code CS is obsolete: Use either the Czech Republic country code CZ, or the Slovakia country code SK.

Country code DE was used for the former Federal Republic of Germany.

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 29. 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. YY assumes range set by CEESCN.</td>
</tr>
<tr>
<td>YY</td>
<td>2-digit year</td>
<td>00-99</td>
<td>Y and ZY valid only if used with &lt;JJJJ&gt;, &lt;CCCC&gt;, or &lt;CCCCCC&gt;.</td>
</tr>
<tr>
<td>YYY</td>
<td>3-digit year</td>
<td>000-999</td>
<td></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') Showa (X'0E45B3457A0F') Taisho (X'0E455B45770F') Meiji (X'0E45A645840F')</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 30 on page 520.</td>
</tr>
<tr>
<td>&lt;CCCC&gt; &lt;CCCCCCC&gt;</td>
<td>Era name in DBCS characters</td>
<td>(X'0E4D8256CE0F') (X'0E4C845ADD4D8256CE0F')</td>
<td>Affects YY field: if &lt;CCCC&gt; specified, YY means the year within the era. See example in Table 30 on page 520.</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>Ibbb-Xllb (Left justified)</td>
<td>For input, source string is folded to uppercase. For output, uppercase only. In=Jan, II=Feb, ..., XII=Dec.</td>
</tr>
<tr>
<td>MMM</td>
<td>3-char month, uppercase</td>
<td>JAN-DEC</td>
<td>For input, output string always folded to uppercase. For output, M generates uppercase and n generates lowercase. Output is padded with blanks (b) (unless 7 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>Mmmmm...m</td>
<td>3-20 char mo., mixed case</td>
<td>January-December</td>
<td></td>
</tr>
<tr>
<td>MMMMM...M</td>
<td>3-20 char mo., uppercase</td>
<td>JANUARY-DECEMBER</td>
<td></td>
</tr>
<tr>
<td>MMMMMMMMMMZ</td>
<td>trailing blanks suppressed</td>
<td>Januarytbb-Decemerb</td>
<td></td>
</tr>
<tr>
<td>Mmmmmmmmmzz</td>
<td>trailing blanks suppressed</td>
<td>Januaryybb-Decemebt</td>
<td></td>
</tr>
<tr>
<td>DD ZD DDD</td>
<td>2-digit day of month 1- or 2-digit day of mo. 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>MI</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 999999</td>
<td>tenths of a second hundreds of a second thousandths of a second</td>
<td>0-9 00-99 000-999</td>
<td>No rounding.</td>
</tr>
<tr>
<td>AP aP a.P. a.p.</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, AP generates uppercase and aP generates lowercase.</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, is are ignored. For output, it generates uppercase and w generates lowercase. Output padded with blanks (unless z specified) or truncated to match the number of is, 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>WWWWWWWW...W</td>
<td>3-20 char day, uppercase</td>
<td>SUNDAY-SATURDAY</td>
<td></td>
</tr>
<tr>
<td>WWWWWWWWWWZ</td>
<td>3-20 char day, mixed case</td>
<td>Sunday-Saturday</td>
<td></td>
</tr>
<tr>
<td>WWWWWWWWW</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 29. 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 30 for an example.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 30. 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>YYYMMDD</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.</td>
</tr>
<tr>
<td>&lt;JJJJ&gt; YY.MM.DD</td>
<td>Showa 63.05.16</td>
<td></td>
</tr>
<tr>
<td>MMDDYY</td>
<td>050688</td>
<td>Accepts imbedded blanks 1-digit year format (?) valid for output only</td>
</tr>
<tr>
<td>MM/DD/YY</td>
<td>05 688</td>
<td></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>Z suppresses zeros/blanks</td>
</tr>
<tr>
<td>DD MMM YY</td>
<td>09 JUN 88</td>
<td></td>
</tr>
<tr>
<td>ZD MMMMmmmmm YY</td>
<td>09 June 88</td>
<td></td>
</tr>
<tr>
<td>MMMMmmmmmzz ZD, YYYY</td>
<td>June 9, 1988</td>
<td></td>
</tr>
<tr>
<td>YY.DDD</td>
<td>88.137</td>
<td>Julian date</td>
</tr>
<tr>
<td>YYDDD</td>
<td>88137</td>
<td></td>
</tr>
<tr>
<td>YYY/DDD</td>
<td>1988/137</td>
<td></td>
</tr>
<tr>
<td>YYYMDHHMISS</td>
<td>880516204229</td>
<td>Timestamp valid only for CEESECS and CEEDATE. If used with CEEDATE, time positions are left blank. If used with CEEDAYS, HH, MI, SS, and 999 fields are ignored.</td>
</tr>
<tr>
<td>YYYYMDHHMISS</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>Wwwwwwwwwwz DD MMM YYYY ZH:MI AP</td>
<td>Monday, 16 May 1988, 8:42 PM</td>
<td></td>
</tr>
</tbody>
</table>

Note: Lowercase characters must be used only for alphabetic picture terms.

Table 31. 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'0E458D45BA0F'</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" on page 525.

The following run-time options control storage allocation:

- ANYHEAP
- BELOWHEAP
- HEAP
- HEAPPOOLS
- LIBSTACK
- STORAGE
- STACK
- THREADHEAP
- THREADSTACK

Ensure that these options are tuned appropriately to avoid performance problems. Tuning tips are provided in the z/OS Language Environment Programming Guide. For an example and complete description of the storage report, see z/OS Language Environment Debugging Guide.

The statistics for initial and incremental allocations of storage types that have a corresponding run-time option differ from the run-time option settings when their values have been rounded up by the implementation, or when allocations larger than the amounts specified were required during execution. All of the following are rounded up to an integral number of double-words:

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

### Stack storage statistics

Language Environment stack storage is managed at the thread level—each thread has its own stack-type resources.

**STACK, THREADSTACK, and LIBSTACK statistics**

- Initial size—the actual size of the initial segment assigned to each thread. (If a pthread-attributes-table is provided on the invocation of pthread-create, then the stack size specified in the pthread-attributes-table will take precedence over the STACK run-time option.)
Increment size—the size of each incremental segment acquired, as determined by the increment portion of the corresponding run-time option.

Maximum used by all concurrent threads—the maximum amount allocated in total at any one time by all concurrently executing threads.

Largest used by any thread—the largest amount allocated ever by any single thread.

Number of segments allocated—the number of incremental segments allocated by all threads.

Number of segments freed—the number of incremental segments freed by all threads.

The number of incremental segments freed could be less than the number allocated if any of the segments were not freed individually during the life of the thread, but rather were freed implicitly in the course of thread termination.

**Determining the applicable threads**

If the application is not a multithreading or PL/I multitasking application, then the STACK statistics are for the one and only thread that executed, and the THREADSTACK statistics are all zero.

If the application is a multithreading or PL/I multitasking application, and THREADSTACK was not suppressed, then the STACK statistics are for the initial thread (the IPT), and the THREADSTACK statistics are for the other threads. However, if THREADSTACK was suppressed, then the STACK statistics are for all of the threads, initial and other.

**Allocating stack storage**

Another type of stack, called the reserve stack, is allocated for each thread and used to handle out-of-storage conditions. Its size is controlled by the 4th subparameter of the STORAGE run-time option, but its usage is neither tracked nor reported in the storage report.

In a multithreaded environment, including PL/I multitasking applications, the initial allocations for all types of stack are made from library heap storage. Library stack and reserve stack are always allocated from BELOWHEAP. User stack is allocated from BELOWHEAP if STACK(,,BELOW) is specified or if ALL31(OFF) is specified; otherwise, it is allocated from ANYHEAP.

Language Environment acquires some initial storage, which is neither stack nor heap, at thread creation time; this storage is allocated from BELOWHEAP if ALL31(OFF) is in effect, or from ANYHEAP if ALL31(ON) is in effect.

**Heap storage statistics**

Language Environment heap storage, other than what is explicitly defined using THREADHEAP, is managed at the enclave level—each enclave has its own heap-type resources, which are shared by the threads that execute within the enclave. Heap storage defined using THREADHEAP is controlled at the thread level.

**HEAP, HEAP24, THREADHEAP, ANYHEAP, and BELOWHEAP statistics**

- Initial size—the default initial allocation, as specified by the corresponding run-time option. For HEAP24, the initial size indicates the value specified for the initsz24 suboption of the HEAP option.
• Increment size—the minimum incremental allocation, as specified by the corresponding run-time option. For HEAP24, the increment size is the value of the incrsz24 suboption of the HEAP option.

**THREADHEAP statistics**
• Maximum used by all concurrent threads—the maximum total amount used by all concurrent threads at any one time.
• Largest used by any thread—the largest amount used by any single thread.

**HEAP, HEAP24, ANYHEAP, BELOWHEAP, and additional heap statistics**
• Total heap storage used—the largest total amount used by the enclave at any one time.

**Additional heap statistics**
Besides the fixed types of heap, additional types of heap can be created, each with its own heap ID. You can create and discard these additional types of heap by using the CEECRHP
• Successful Create Heap requests—the number of successful Create Heap requests.
• Successful Discard Heap requests—the number of successful Discard Heap requests.

The number of Discard Heap requests could be less than the number of Create Heap requests if the special heaps allocated by individual Create Heap requests were not freed individually, but rather were freed implicitly in the course of enclave termination.

**Heap pools storage statistics**
The HEAPPOOLS run-time option controls usage of the heap pools storage algorithm at the enclave level. The heap pools algorithm allows for the definition of one to six heap pools, each consisting of a number of storage cells of a specified length.
Heap pools statistics

- Pool size: `ssss`
  - `p` — the number of the pool
  - `ssss` — the cell size specified for the pool.
- Successful Get Heap requests: `xxxx-yyyy n`
  - `xxxx` — the low side of the 8 byte range
  - `yyyy` — the high side of the 8 byte range
  - `n` — the number of requests in the 8 byte range.
- Requests greater than the largest cell size — the number of storage requests that are not satisfied by heap pools.

Note: Values displayed in the heap pools statistics report are not serialized when collected, therefore the values are not necessarily exact.

Heap pools summary

The Heap pools summary displays a report of the heap pool statistics and provides suggested percentages for current cell sizes as well as suggested cell sizes.

- Cell Size — the size of the cell specified in the HEAPPOOLS run-time option
- Extent Percent — the cell pool percent specified by the HEAPPOOLS run-time option
- Cells Per Extent — the number of cells per extent. This number is calculated using the following formula:
  \[ \text{Initial Heap Size} \times \left( \frac{\text{Extent Percent}}{100} \right) \div (8 + \text{Cell Size}) \]

  with a minimum of one cell.

Note: Having only one cell per extent is not recommended since the pool could allocate many extents, which would cause the heap pool algorithm to perform inefficiently.

- Extents Allocated — the number of times that each pool allocated an extent.
  In order to optimize storage usage, the extents allocated should be either one or two. If the number of extents allocated is too high, then increase the percentage for the pool.
- Maximum Cells Used — the maximum number of cells used for each pool.
- Cells In Use — the number of cells that were never freed.

Note: A large number in this field could indicate a storage leak.

- Suggested Percentages for current Cell Sizes — percentages calculated to find the optimal size of the cell pool extent. The calculation is based on the following formula:
  \[ \left( \frac{\text{Maximum Cells Used} \times (\text{Cell Size} + 8) \times 100}{\text{Initial Heap Size}} \right) \]
  With a minimum of 1% and a maximum of 90%

  Make sure that your cell pool extents are neither too large nor too small. If your percentages are too large then additional, unreferenced virtual storage will be allocated, thereby causing the program to exhaust the region size. If the percentages are too small (only one cell per extent — the worst case scenario) then the heap pools algorithm will run inefficiently.

- Suggested Cell Sizes: — sizes that are calculated to optimally use storage (assuming that the application will malloc/free with the same frequency).
**Note:** The suggested cell sizes are given with no percentages because the usage of each new cell pool size is not known. If there are less than 6 pool sizes calculated then the last pool will be set at 2048.


---

### Storage statistics for AMODE 64 applications

The following run-time options control storage allocation:

- HEAP64
- HEAPPOOLS64
- IOHEAP64
- LIBHEAP64
- STACK64
- THREADSTACK64


The statistics for initial and incremental allocations of storage types that have a corresponding run-time option differ from the run-time option settings when their values have been rounded up by the implementation, or when allocations larger than the amounts specified were required during execution. See the descriptions of the run-time options for information about rounding.

### Stack storage statistics for AMODE 64 applications

Language Environment stack storage is managed at the thread level—each thread has its own stack-type resources.

**STACK64 and THREADSTACK64 statistics**

- **Initial size**—the actual size of the initial stack area assigned to each thread. If a pthread-attributes-table is provided on the invocation of pthread-create, the stack size specified in the pthread-attributes-table takes precedence over the stack run-time options.
- **Increment size**—the size of each incremental stack area made available, as determined by the increment portion of the corresponding run-time option.
- **Maximum used by all concurrent threads**—the maximum amount allocated in total at any one time by all concurrently executing threads.
- **Largest used by any thread**—the largest amount allocated ever by any single thread.
- **Number of increments allocated**—the number of incremental segments allocated by all threads.

**Determining the applicable threads**

If the application is not a multithreading application, the STACK64 statistics are for the one and only thread that executed, and the THREADSTACK64 statistics are all zero.

If the application is a multithreading application, and THREADSTACK64 was not suppressed, the STACK64 statistics are for the initial thread (IPT), and the
THREADSTACK64 statistics are for the other threads. However, if THREADSTACK64 was suppressed, the STACK64 statistics are for all of the threads, initial and other.

**Allocating stack storage**
The allocation of the stack for each thread, including the initial processing thread (IPT), is part of a storage request to the system when the thread is first created. Other storage, not part of the stack, is also acquired at this time. These storage allocations are not shown in the storage report. The size of the stack portion of this storage is the stack maximum size plus a one megabyte (1M) guard area. After allocation, the guard area follows the stack initial size and runs through the end of the stack maximum size plus the 1M guard area. Increments to the stack for each thread do not result in additional storage requests to the system. They result in the movement of the beginning of the guard area no further than the maximum size of the stack. The stack initial, increment, and maximum sizes are controlled through the STACK64 and THREADSTACK64 run-time options.

**Heap storage statistics**
Language Environment heap storage is managed at the enclave level. Each enclave has its own heap type resources, which are shared by the threads that execute within the enclave. The heap resources have 64-bit, 31-bit, and 24-bit addressable areas, each of which can be tuned separately.

**HEAP64, LIBHEAP64, and IOHEAP64 statistics**
- Initial size—the default initial allocation, as specified by the corresponding run-time option.
- Increment size—the minimum incremental allocation, as specified by the corresponding run-time option.
- Total heap storage used—the largest total amount used by the enclave at any one time.
- Successful Get Heap requests—the number of get heap requests.
- Successful Free Heap requests—the number of free heap requests.
- Number of segments allocated—the number of incremental segments allocated.
- Number of segments freed—the number of incremental segments individually freed.

The number of Free Heap requests could be less than the number of Get Heap requests if the pieces of heap storage acquired by individual Get Heap requests were not explicitly freed, but were freed implicitly during enclave termination. The number of incremental segments individually freed could be less than the number allocated if the segments were not explicitly freed, but were freed implicitly during enclave termination. The initial segment is included in Number of segments allocated for each 31-bit and 24-bit addressable heap resource, and for the 64-bit addressable IOHEAP64 resource. A disposition of KEEP always causes 0 to be reported for the Number of segments freed. These statistics, in all cases, specify totals for the entire enclave.

**Heap pools storage statistics**
The HEAPPOOLS64 run-time option controls usage of the heap pools storage algorithm at the enclave level. The heap pools algorithm allows for the definition of one to twelve heap pools, each consisting of a number of storage cells of a specified length.
Heap pools statistics

- Pool \( p \) size: \( ssss \)
  - \( p \) — the number of the pool
  - \( ssss \) — the cell size specified for the pool.
- Successful Get Heap requests: \( xxxx-yyyy \ n \)
  - \( xxxx \) — the low side of the 8 byte range
  - \( yyy \) — the high side of the 8 byte range
  - \( n \) — the number of requests in the 8 byte range.
- Requests greater than the largest cell size — the number of storage requests that are not satisfied by heap pools.

Note: Values displayed in the heap pools statistics report are not serialized when collected, therefore the values are not necessarily exact.

Heap pools summary

The heap pools summary displays a report of the heap pool statistics and provides suggested cell counts for current cell sizes as well as suggested cell sizes.

- Cell Size — the size of the cell specified in the HEAPPOOLS64 run-time option
- Cells Per Extent — the cell pool count specified by the HEAPPOOLS64 run-time option
- Extents Allocated — the number of times that each pool allocated an extent.
  
  In order to optimize storage usage, the extents allocated should be either one or two. If the number of extents allocated is too high, increase the cell count for the pool.
- Maximum Cells Used — the maximum number of cells used for each pool.
- Cells In Use — the number of cells that were never freed.

Note: A large number in this field could indicate a storage leak.

- Suggested Cell Sizes: — sizes that are calculated to optimally use storage (assuming that the application will malloc/free with the same frequency).

Note: The suggested cell sizes are given with no cell counts because the usage of each new cell pool size is not known. If there are less than 12 pool sizes calculated then the last pool size is set at 65536.

For more information about stack and heap storage for AMODE 64 applications, see

\underline{z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode}
Appendix D. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User's Guide, and z/OS ISPF User's Guide 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/Library Server versions of z/OS books in the Internet library at:

www.ibm.com/servers/eserver/zseries/zos/bkserv/

One exception is command syntax that is published in railroad track format; screen-readable copies of z/OS books with that syntax information are separately available in HTML zipped file form upon request to mhvrdfs@us.ibm.com.
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.
If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**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**

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

- AD/Cycle
- AIX
- AS/400
- BookManager
- C/370
- CICS
- COBOL/370
- DB2
- IBM
- IBMLink
- IMS
- Language Environment
- MVS
- MVS/ESA
- OS/2
- OS/390
- Resource Link
- S/370
- SAA
- VisualAge
- z/OS
- z/OS.e
- zSeries

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

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

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

/ (slash)
   NOEXECOPS alters behavior of 30
   specifying in parameters of TEST run-time
   option 78

* (asterisk) 77, 79
argv parameter for C
   ARGPARSE run-time option’s effect on 18
argc parameter for C
   ARGPARSE run-time option’s effect on 18
calloc() function 33
fprintf function 133
setlocale() function
   CEE3CTY callable service and 122
   CEE3LNG callable service and 166
   COUNTRY run-time option and 23

= (equal sign)
   in ENVAR run-time option 26

' (straight single quote)
   initializing storage with 69, 71
   specifying in parameters of TEST run-time
   option 78, 79

" (straight double quote)
   initializing storage with 69, 71
   specifying in parameters of TEST run-time
   option 78, 79

Numerics

0 type_of_move 383, 387
1 type_of_move 383, 387
16M line
   ALL31 run-time option and 15
   allocating library heap storage not restricted to
   below 16, 17
   ANYHEAP run-time option and 16, 17
   HEAP run-time option and 34
   storage received from CEEGTST and 320
2-digit years
   querying within 100-year range (CEEQCEN) 407
   examples of 408
   setting within 100-year range (CEESCEN) 422
   examples of 423
24-bit addressing
   ANYHEAP run-time option and 17

A

abbreviating run-time options 11
abcode parameter 11, 12, 113
abend codes
   abend 4088, reason code 1004 70
   abend 4091, reason code 21 25
   dumping 113
   ERRCOUNT run-time option and 450
   in CEE3DMP callable service 131
   percolating in ABPERC run-time option 11

abends
   bad filenames in CEEBLDTX macro can cause 401
   dumps, getting when enclave terminates with abend
   (CEE3ABD) 113
   ERRCOUNT run-time option and 28
   in CEEFRST callable service 284
   in CEEGTST callable service 320
   not handled if initiated by SVC 13 87
   options report not generated (RPTOPTS run-time
   option) 61
   out-of-storage condition, cause 70
   percolating
      ABPERC run-time option and 12
      CEEBXITA and 87
   percolating certain abends in CEEBXITA 86
   side effects if encountered using TRAP(OFF) 87
   storage report not generated (RPTSTG run-time
   option) 62
   system abends
      ABPERC run-time option and 12
      percolating in the assembler user exit 87
      terminating enclave with (CEE3ABD) 113
      trapping 86
      using ABPERC run-time option to percolate 12
   when nested condition exceeded (DEPTHCONDLMT
   run-time option) 25
   ABPERC run-time option
      CICS ignores this option 12
      S0Cx abends can’t be percolated 12
      syntax 11
   absolute value math service (CEESxABS) 476
   ABTERMENC run-time option
      abend codes and 12
      choosing between abend codes and return
      codes 12
      syntax 12
   access method service (AMS) 13
   accessibility 529
   active condition
      CEEGPID callable service and 304
      dumping information related to 131
      TERMTHDACT generates message for 72
   additional heap
      creating (CEECRHP) 215
      discarding (CEEDSHP) 256
      reducing the amount of storage you need to run an
      application 283
   address parameter
      CEECZST callable service and 220
      CEEFRST callable service and 284
      CEEGTST callable service and 321
   addressing exception 192
   aggregate
      dumping arrays and structures 129
      AIXBLD run-time option
      syntax 13
alignment
PL/I structures and Language Environment storage callable services 322
ALL31 run-time option
ALL31(ON) default under CICS 16
syntax 14
allocating storage
additional heap, setting size of (CEECRHP) 215
anywhere heap, setting size of (ANYHEAP run-time option) 16, 17
below heap, setting size of (BELOWHEAP run-time option) 19, 20
gett heap storage (CEEGSTST) 321, 323, 325
initial heap, setting size of (HEAP run-time option) 33, 44
initializing when allocating (STORAGE run-time option) 69, 71
library stack storage, setting size of (LIBSTACK run-time option) 46
stack storage, setting size of (STACK run-time option) 65, 68, 69, 84
alternate indices, building with AIXBLD | NOAIXBLD run-time option 14
AMODE
ALL31 run-time option and 14, 15
callable services and 104
heap storage 35
STACK run-time option and 15
under CICS 322
AMS (access method service) 13
ANYHEAP run-time option
syntax 16
anywhere heap
allocating storage from 16, 17
ANYHEAP run-time option and 16, 17
controlling whether storage is freed from 17
ANYWHERE suboption
ANYHEAP run-time option and 17
HEAP run-time option and 34
application writer interface (AWI) 103
arccosine math service (CEESxACS) 476
arcsine math service (CEESxASN) 477
arctangent math service (CEESxATN) 479
arctangent2 math service (CEESxAT2) 480
AREA storage for PL/I 322
ARGPARSE run-time option
CICS ignores this option 18
syntax 17
argument
dumping 129
format of in invoked routine 55
list format
effect of EXECOPS run-time option on 30
arithmetic (continued)
calculation on dates (continued)
convert date to number of seconds (CEESECS) 435, 438
get current Greenwich Mean Time (CEEGMT) 296, 297
examples using 474, 510, 512
parameter types 473
services
Quick Reference Tables 100, 101
text variations of 100, 101, 473
array 129
assembler language
routines
calling math services from 473
invoking callable services from 105
asterisk (*) 77, 79
AT OCCURRENCE debug command for Debug Tool 77
AT TERMINATION debug command for Debug Tool 77
attention processing
handling with a debug tool (TEST run-time option) 77
handling with INTERRUPT run-time option 43
AUTOTASK run-time option
syntax 18
B
BASED storage for PL/I 33
below heap
allocating storage from 19, 20
BELOWHEAP run-time option and 19, 20
controlling whether storage is freed 19
BELOW suboption
ANYHEAP run-time option and 17
BELOWHEAP run-time option
syntax 19
BIF (built-in functions), in CEE3DMP 131
bimodal addressing
ANYHEAP run-time option 17
HEAP run-time option 34
binary-valued flags 89
bit manipulation services 469
CEESICLR—bit clear 469
CEESISET—bit set 469
CEESISHF—bit shift 470
CEESITST—bit test 470
buffer
dumping contents of buffers used by files 129
storing messages in examples of 368
syntax description 364
building condition token
examples of 403, 406
syntax description 400
built-in functions (BIF) for PL/I, in CEE3DMP 127
call return point 382
CALL statement for PL/I
fetchable main
CEE3DMP ENCLAVE considerations 131
callable services
AMODE switching across calls to 15
CEE3ABD—terminate enclave with an abend 113
CEE3CIB—return pointer to condition information block 115
CEE3CTY—set default country 120
CEE3DMP—generate dump 127
CEE3GRC—get the enclave return code 135
CEE3GRN—get name of routine that incurred condition 148
CEE3GRO—get offset of condition 156
CEE3LNG—set national language 163
CEE3MCS—get default currency symbol 172
CEE3MDS—get default decimal separator 175
CEE3MTS—get default thousands separator 179
CEE3PRM—query parameter string 183
CEE3RPH—set report heading 186
CEE3SPM—query and modify Language Environment hardware condition enablement 189
CEE3SRC—set the enclave return code 196
CEE3SRP—set resume point 197
CEE3USR—set or query user area fields 198
CEE3ZST—reallocate storage 223
CEECHDLR—register user-written condition handler 137, 150, 327
CEECHDLU—unregister user-written condition handler 337
CEEHDLR—register user-written condition handler 137, 150, 327
CEEHDLU—unregister user-written condition handler 337
CEEISEC—convert integers to seconds 344
CEEITOK—return initial condition token 350
CEEJLTD—get current local date or time 362
CEEMGET—get a message 368
CEEMOUT—dispatch a message 373
CEEMRCR—move resume cursor 388
CEEMSG—get, format, and dispatch a message 396
CEEENCOD—construct a condition token 403
CEEHEX—get hex digit from condition token 408
CEEERAN0—calculate uniform random numbers 419
CEESCEN—set century window 423
CEESDLG1—logarithm base 10 510
CEESEC—convert seconds to integers 432
CEESECS—convert timestamp to seconds 439
CEESSGL—signal a condition 150, 452
CEESIMOD—modular arithmetic 510
CEETEST—invoke debug tool 464
fc parm, omitting 107
hardware interrupts that cannot be enabled 192
HEAP considerations 35
invoking callable services from 107
mapping SPIE and STAE to TRAP 87
MSGFILE run-time option and 49
perror() function 49
PLIST setting under IMS differs depending on version of C compiler 55
run-time options specific to Language Environment
ARGPARSE 17
ENV 26
EXECOPS 29
PLIST 55
REDIR 59
run-time options, specifying on the command line 29
stderr
MSGFILE run-time option and 49
REDIR run-time option and 59
redirecting output from 49
variables, where stored 33
"
callable services  (continued)
CEECLBDY—convert date to COBOL Lilian format 203
CEEClMI—store and load message insert data  208
CEEClRHP—create new additional heap 215
CEEClSTZ—reallocate (change size of) storage 220
CEEDATE—convert Lilian date to character format 227
CEEDATM—convert seconds to character timestamp 234
CEEDAYS—convert date to Lilian format 242
CEEDECOD—decompose a condition token 249
CEEDSHP—discard heap 256
CEEDYWK—calculate day of week from Lilian date 261
CEEFMDA—get default date format 267
CEEFMDT—get default date and time format 270
CEEFMON—format monetary string 274
CEEFMTM—get default time format 280
CEEFRST—free heap storage 283
CEEFTDS—format date and time into character string 289
CEEGMT—get current Greenwich Mean Time 296
CEEGLPI—get offset from Greenwich Mean Time to local time 300
CEEISPID—retrieve Language Environment version and platform ID 304
CEEQDV—retrieve q_data_token 309
CEEGLST—get heap storage 320
CEEHLDR—register user condition handler 325
CEEHDLU—unregister user condition handler 335
CEEISEC—convert integers to seconds 341
CEEITOK—return initial condition token 348
CEEJCNV—query locale numeric conventions 354
CEEJCT—get current local time 360
CEEJMET—get a message 364
CEEJMT—dispatch a message 371
CEEJMR—move resume cursor explicit 376
CEEJMRQ—move resume cursor relative to handle cursor 382
CEEJMSG—get, format, and dispatch a message 394
CEEJNCOD—construct a condition token 400
CEEKCN—query the century window 407
CEEKTC—query locale date and time conventions 410
CEEKRYL—query active locale environment 416
CEEKANO—calculate uniform random numbers 418
CEECSCN—set the century window 421
CEESCL—compare string collation weight 426
CEESCR—convert seconds to integers 430
CEESSCS—convert timestamp to number of seconds 435
CEESETL—set the locale operating environment 443
CEESSGL—signal a condition 449
CEESISCL—bit clear 469
CEESISET—bit set 469
CEESISHS—bit shift 470
CEESIST—bit test 470
CEESTXF—transform string characters into collation weights 455
CEESAXBS—absolute value 475
CEESAXCS—arccosine 476
CEESAXSN—arc sine 477
CEESAXAT—artangent of two arguments 480
CEESAXTH—hyperbolic arctangent 478
CEESAXTN—arctangent 479
CEESAXCJG—hyperbolic conjugate complex 480
CEESAXCOS—cosine 481
CEESAXCOSH—hyperbolic cosine 483
CEESAXCTN—cotangent 484
CEESAXDIM—positive difference 486
CEESAXDV—division 487
CEESAXER—error function 487
CEESAXERF—error function complement 488
CEESAXEXP—exponent (base e) 489
CEESAXGMA—gamma function 490
CEESAXIM—imaginary part of complex 491
CEESAXINT—truncation 492
CEESAXLG1—logarithm base 10 494
CEESAXLG2—logarithm base 2 495
CEESAXLG—log gamma function 493
CEESAXLOG—logarithm base e 496
CEESAXMLT—floating complex multiply 497
CEESAXMOD—modular arithmetic 497
CEESAXNIN—nearest integer 499
CEESAXWNW—nearest whole number 499
CEESAXSGN—transfer of sign 500
CEESAXSIN—sine 501
CEESAXSNH—hyperbolic sine 503
CEESAXSQT—square root 504
CEESAXTAN—tangent 505
CEESAXTH—hyperbolic tangent 507
CEESAXTP—exponential (* *) 507
CEETDLI—invoke IMS 459
CEETEST—invoke debug tool 462
data types allowed in 110, 113
feedback code parameter 109
involving 105
in C 104, 107
in COBOL 107, 108
in PL/I 108, 110
Quick Reference Tables 97, 100
case 1 condition token 402
case 2 condition token 402
cause code condition 402
CBLOPTS run-time option syntax 20
CBLPSHPOP run-time option
EXEC CICS PUSH and EXEC CICS POP commands and 21
syntax 20
CBLQDA run-time option
CICS ignores this option 22
relationship to dump and message file 21
syntax 21
CEE_ENTRY Language Environment data type and HLL equivalents 110
CEE3 prefix, meaning of 104
CEE3ABD—terminate enclave with an abend
  CICS considerations  113
  examples using  114, 115
  syntax  113
CEE3CIB—return pointer to condition information
  block  115
  syntax  115
CEE3CTY—set default country
  COUNTRY run-time option and  23
  date and time services and  120
  examples using  123, 126
  other national language support services and  120
  RPTOPTS run-time option and  120
  RPTSTG run-time option and  120
  setlocale() and  122
  syntax  120
  table of default values for specified
    country_code  515
CEE3DMP—generate dump
  default dump file of  127
  examples using  133, 135
  options used in TERMTHDACT run-time option  76
  syntax  127
  using TEST compile-time option with  129
CEE3GRC—get the enclave return code
  examples using  137, 147
  syntax  135
CEE3GRN—get name of routine that incurred condition
  examples using  150, 156
  syntax  148
CEE3GRO—get offset of condition
  examples using  158
  syntax  156
CEE3LNG—set national language
  default if invalid national language specified  164
  examples using  167, 171
  messages and  163
  NATLANG run-time option and  51
  setlocale() and  166
  syntax  163
CEE3MCS—set default currency symbol
  CEE3CTY callable service  172
  COUNTRY run-time option and  172
  default if invalid country_code specified  173
  examples using  173, 175
  syntax  172
  table of default values for specified
    country_code  515
CEE3MDS—set default decimal separator
  CEE3CTY callable service and  176
  COUNTRY run-time option and  176
  default if invalid country_code specified  176
  defaults of country_code parameter  515
  examples using  177, 179
  syntax  175
CEE3MTS—set default thousands separator
  CEE3CTY callable service and  180
  COUNTRY run-time option and  180
  default if invalid country_code specified  180
  examples using  181
  syntax  179
CEE3MTS—get default thousands separator
  (continued)
  table of default values for specified
    country_code  515
CEE3PRM—query parameter string  407
  examples using  184, 186
  syntax  183
CEE3RPH—set report heading
  examples using  186
  syntax  186
CEE3SPM—query and modify Language Environment
  hardware condition enablement
  examples using  193, 195
  syntax  189
CEE3SRC—set the enclave return code
  examples using  197
  syntax  196
CEE3SRP—set resume point
  examples using  198
  syntax  197
CEE3USR—set or query user area fields
  examples using  201, 203
  syntax  198
CEEBLDTX EXEC
  use caution when creating new facility ID for  401
  CEEBXITA assembler user exit
  percolating certain abends with TRAP(ON)  87
CEECBLDY—convert date to COBOL Lilian
  format  203
  examples using  204
  syntax  203
CEECMI—store and load message insert data  208
  examples using  211, 214
  syntax  208
CEECRHP—create new additional heap
  examples using  218, 220
  HEAP run-time option and  321
  syntax  215
CEECZST—reallocate (change size of) storage
  CEEGTST callable service and  221
  examples using
    example with CEEGTST and CEEFRST  223
    examples with CEEHDLR, CEEGTST, and
    CEEEMR CR  223, 226
  syntax  221
CEEDATE—convert Lilian date to character format
  CEE DAYS callable service and  227
  CEEFMDA callable service and  228
  COUNTRY run-time option and  228
  examples using  230, 234
  syntax  227
  table of sample output  234
CEEDATM—convert seconds to character timestamp
  CEEFMDT callable service and  235
  CEESECI callable service and  430
  CEESECS callable service and  234
  COUNTRY run-time option and  235
  examples using  237, 242
  syntax  234
  table of sample output  242
CEEDAYS—convert date to Lilian format
CEEDATE callable service and 242
CEESCEN callable service and 203, 243
examples using 249
syntax 242
CEEDCOD—decompose a condition token
examples using
C example showing alternative to using 452
examples with CEEGTST 252, 255
syntax 249
CEEDOPT
CBOLOTS run-time option must be specified in
CEEDOPT or CEEUOPT 20
COUNTRY run-time option consideration 23
country_code and 23
ENVAR run-time option and 27
national language codes and 51, 164
 specifying nonexistent national language code in 51
 specifying UPSI run-time option in 89
CEEDSHP—discard heap
 can overrule HEAP run-time option 257
CEECHRHP callable service and 256
examples with CEECHRHP 258, 260
HEAP run-time option and 257
syntax 256
CEEDUMP default dump file
CBLQDA run-time option and 21
CEE3DMP callable service and 127
CEEDYWK—calculate day of week from Lilian date
examples using 263, 266
syntax 261
CEEEBMAW file, description of 105
CEEEDCCT file, description of 105
CEEFMDA—get default date format
CEE3CTY callable service 267
COUNTRY run-time option and 267
default if invalid country_code specified 267
defaults of country_code parameter 515
examples using 268, 270
syntax 267
CEEFMDT—get default date and time format
CEE3CTY callable service and 271
COUNTRY run-time option and 271
default if invalid country_code specified 271
defaults of country_code parameter 515
examples using 272, 274
syntax 270
CEEFMON—format monetary string
about 274
examples using 277
CEEFMTM—get default time format
CEE3CTY callable service and 280
COUNTRY run-time option and 280
default if invalid country_code specified 280
defaults of country_code parameter 515
examples using 281, 283
syntax 280
CEEFRST—free heap storage (continued)
HEAP run-time option and 284
syntax 283
CEEFTZDS—format date and time into character string
syntax 289
CEEGMT—get current Greenwich Mean Time 467
CEEDATE callable service and 297
CEEDATM callable service and 297
CEEGMTO callable service and 296
examples using 298, 300
syntax 296
CEEGMTO—get offset from Greenwich Mean Time to
local time
CEEDATM callable service and CEEDATM and 301
examples using 303, 304
syntax 300
CEEGPID—retrieve the Language Environment version
and platform ID
examples using 306, 309
syntax 304
CEEGQDT—retrieve q_data_token
CEE3SGL callable service and 309
examples using
 CEEQDT by itself 312
examples with CEEHDLR and CEE3SGL 312
instance specific information (ISI) and 309
syntax 309
CEEGTST—get heap storage
CEECHRHP callable service and 321
CEEDSHP callable service and 320
CEEFRST callable service and 320
CEE3SGL callable service and 320
examples using
 examples with CEECZST and CEEFRST 223
examples with CEEFRST 286, 323, 325
HEAP run-time option and 321
syntax 320
using STORAGE run-time option to initialize storage
 received from 320
CEEHDLR—register user condition handler
CEE3SRC and CEE3GRC 150, 156
condition handling example 327
examples using
 CEEHDLR by itself 327, 334
examples with CEE3SRC and CEE3GRC 150, 156
syntax 325
CEEHDLU—unregister user condition handler
CEEHDLR callable service and 336
examples with CEEHDLR 337, 340
syntax 335
CEEIBMCI file, description of 105
CEEIBMCT file, description of 105
CEEITOK—return initial condition token
examples using 350, 354
CEEITOK—return initial condition token  (continued)
syntax  348
CEELCNV—query locale numeric conventions
syntax  354
CEEOCT—get current local time
CEEDATM callable service and  361
CEEGMT callable service and  361
CEEGMTO callable service and  361
examples using  362, 364
syntax  360
CEEMGET—get a message
examples using
CEEEMGET by itself  368, 371
examples with CEEMOUT  368
syntax  364
CEEMOUT—dispatch a message
examples using  373, 375
MSGFILE run-time option and  372
syntax  371
CEEMRCE—move resume cursor explicit
examples using  380, 381
syntax  376
CEEMRCR—move resume cursor relative to handle
cursor
examples with CEEHDLR and CEESGL  388, 393
illustrations of  385
syntax  382
CEEMSG—get, format, and dispatch a message
examples using  396, 399
MSGFILE run-time option and  395
MSGQ run-time option and  51
syntax  394
CEENCOD—construct a condition token
CEEDCOD callable service and  249
examples using  403, 406
how C users can use CEESGL callable service
instead of  452
syntax  400
CEEQCEN—query the century window
CEESCEN callable service and  407
examples using  408, 410
syntax  407
CEEQDT—query locale, date, and time conventions
syntax  410
CEEQRYL—query active locale environment
syntax  416
CEERANO—calculate uniform random numbers
examples using  419, 421
syntax  418
CEESCEN—set the century window
CEEDAYS callable service and  421
CEESCECS callable service and  421
examples using  423, 426
syntax  421
CEESCOL—compare string collation weight
syntax  426
CEESECI—convert seconds to integers
examples using  432, 435
relationship to CEEISEC callable service  431
syntax  430
CEESCECS—convert timestamp to number of seconds
CEEDATM callable service and  436
CEEISEC callable service and  342
CEESCECS callable service and  421
COUNTRY run-time option and  437
syntax  435
CEESETL—set locale operating environment
syntax  443
CEESGL—signal a condition
CEEGQDT callable service and  309
examples using  452, 455
HLL-specific condition handlers and  450
q_data_token and  309, 450
setting of ERRCOUNT run-time option can cause
abend  450
syntax  449
TRAP run-time option does not affect  86
using to create an ISI  449, 450
CEESTXF—transform string into collation weights
syntax  455
CEETDLI interface to IMS
syntax  459
CEETEST—invoke debug tool
examples using  464, 466
NOTEST run-time option and  79
syntax  462
CEEUOPT
CBLOPTS run-time option must be specified in
CEEUOPT or CEEDOPT  20
COUNTRY run-time option considerations  23
country_code and  23
national language codes and  51, 164
specifying nonexistent national language code in
specifying UPSI run-time option in  24, 89
century window
CEEDAYS callable service and  203, 243
CEEQCEN callable service and  407
examples of  408, 410
CEESCECS callable service and  422
examples of  408, 410, 423, 426
CEESCECS callable service and  436
CESTE transient data queue
CEEMOUT callable service and  372
TERMTHDACT run-time option and  76
car char_parm_string parameter  184
caracter timestamp
converting Lilian seconds to (CEEDATM)  234
examples of  237, 242
converting to COBOL Lilian seconds
(CEECBLDY)  203
examples of  207
converting to Lilian seconds (CEESCECS)
examples of  435
CHARn Language Environment data type and HLL
equivalents  110
CHECK run-time option
syntax  22
using while debugging  22
checking errors, flagging with CHECK run-time
option  22
ChuHwaMinKow era  521

Index  541
CIB (Condition Information Block) 115
CICS
callable service behavior under
  abend codes in CEE3ABD callable service 113
  storage considerations 322
CBLPSHPOP run-time option and 20
CEE3DMP ENCLAVE considerations 131
CESE transient data queue and 49
PLIST setting 54
  storage and 322
class code condition 402
clean-up parameter 113
clearing storage 69
CMS
  CEE3DMP special considerations 131
  CMSSTOR OBTAIN/RELEASE commands 62
OSRUN command 55
  PLIST run-time option settings to specify when
running under 54
  specifying a C application is running under 26, 29
CMSCALL
  CEE3DMP ENCLAVE considerations 131
  CMSSTOR OBTAIN/RELEASE 62
COBOL
  batch debugging features 24
callable services, invoking from 107, 108
declaration for Language Environment data types 110
examples
  CEE3ABD—terminate enclave with abend 114
  CEE3CTY—set default country 123
  CEE3DMP—generate dump 133
  CEE3GRC—get enclave return code 137, 146
  CEE3GRN—get name of routine that incurred
condition 151
  CEE3LNG—set national language 167
  CEE3MCS—get default currency 173
  CEE3MDS—get default decimal separator 177
  CEE3MTS—get default thousands separator 181
  CEE3PRM—query parameter string 184
  CEE3RPH—set report heading 187
  CEE3SMP—query and modify Language
Environment hardware condition
  enablement 193
  CEE3SRC—set enclave return code 137, 146
  CEE3USR—set or query user area fields 201
  CEECMl—store and load message insert
  data 211
  CEECRHP—create new additional heap 218
  CEECZST—reallocate storage 223
  CEEDATE—convert Lilian date to character
  format 230
  CEEDATM—convert seconds to character
  timestamp 238
  CEEDAYS—convert date to Lilian format 207
  CEECDOD—decompose a condition token 252
  CEEFMON—format monetary string 277
  CEEFTDS—format date and time into character
  string 293
  CEEGPID—retrieve Language Environment
  version/platform ID 306
  CEEGQDT—retrieve q_data_token 312
COBOL (continued)
examples (continued)
  CEEGTST—get heap storage 223, 323
  CEEHDLR—register user-written condition
  handler 137, 146, 327
  CEEHDLU—unregister user-written condition
  handler 337
  CEEISIEC—convert integers to seconds 344
  CEEITOK—return initial condition token 350
  CEELCNY—query locale numeric
  conventions 358
  CEELOCT—get current local date or time 362
  CEEMQRT—get a message 368
  CEMOUT—dispatch a message 373
  CEEUHDR—move resume cursor 389
  CEMMSG—get, format, and dispatch a
message 396
  CEECNOD—construct a condition token 403
  CEEEQCN—query century window 408
  CCEEDT—return locale date and time 413
  CEEQRYL—query active locale environment 418
  CEESCN—set century window 423
  CEEISCOL—compare string collation weight 426
  CEESEC—convert seconds to integers 432
  CEEEC—convert timestamp to seconds 439
  CEESNL—set locale operating
  environment 428, 446
  CEEESN—signal a condition 452
  CEESSLOG—calculate logarithm base e 510
  CEESTXF—transform string characters into
  collation weights 457
  CEETEST—invoke a debug tool 464
GOBACK statement
  RTEREUS run-time option and 64
hardware conditions that cannot be enabled
  under 192
options, mapping
  mapping STAE to TRAP 86
reusability of an environment when COBOL is
  main 63
run-time options specific to
  AIBXBLD—invoke AMS 13, 14
  CBLOPTS—specify format of argument string 20
  CBLPSHPOP—control usage of CICS
  commands 20
  CBLQDA—control VSAM dynamic allocation 21
  CHECK—allow for checking errors 22
  DEBUG—activate COBOL batch debugging 24
  FLOW—control OS/VS COBOL FLOW output 33
  RTEREUS—make first COBOL routine
  reusable 63
  SIMVRD—specify VSAM KSRS 64
  UPSI—set UPSI switches 88
severity 0 and 1 conditions, ERRCOUNT run-time
  option and 28
space management tuning table, using HEAP
  run-time option with 35
STOP RUN statement 64
tuning, with HEAP run-time option 35
using VSAM KSRS to simulate variable length
  relative organization data sets 13
COBOL (continued)
variables, where stored 33, 65, 68
command
syntax diagrams 6
command line
parsing of C arguments (ARGPARSE run-time option) 17
specifying whether C redirections are allowed from
(REDIR run-time option) 59
specifying whether run-time options can be specified
on (EXECOPS run-time option) 29
commands_file parameter 78
compatibility
CICS 20
complex numbers
complex number math services
conjugate of complex (CEESxCJG) 480
floating complex divide (CEESxDVDF) 487
floating complex multiply (CEESxMLT) 497
imaginary part of complex (CEESxIMG) 491
cond_rep parameter
CEECMI callable service and 209
CEEGQDT callable service and 310
CEESGL callable service and 450
cond_str parameter 191
cond_token parameter
CEEMGET callable service and 365
CEEMSG callable service and 395
CEENCOD callable service and 402
condition
active
CEEGPID callable service and 304
dumping information related to 130
specifying how much information produced for 76
cause code 402
class code 402
enabled 449
getting name of routine that incurred condition
(CEE3GRN) 148
examples of 150, 156
getting offset of condition 156
initial
allowing the handling of 25
returning initial condition token for (CEEITOK) 348
nested, allowing levels of 24
original 76
promoted 76
providing q_data_token for (in CEESGL)
examples of 452, 455
syntax description 449
safe conditions 450
severity
CEESGL callable service and 450
ERRCOUNT run-time option and 28
severity levels that cause the debug tool to gain
control 77
TERMTHDACT run-time option and 72
tolerating a given number of 25
condition (continued)
unhandled
level of information produced for (TERMTHDACT
run-time option) 72
condition handler
C signal handlers
CEEMRCR callable service and 382
CEESGL callable service and 450
HLL semantics
ABPERC run-time option and 12
TRAP run-time option and 86
Language Environment condition handler 86
PL/I ON-units
CEESGL callable service and 450
user-written
allowing nested conditions in 24
CEESPM callable service and 190
moving the resume cursor from a 382
registering with CEETHDLR 325
retrieving q_data_token 309
TRAP run-time option and 86
unregistering 336
vector facility and 90
condition handling
callable services for
CEE3ABD—terminate enclave with an
abend 113
CEE3GRN—get name of routine that incurred
condition 148
CEE3GRO—get offset of condition 156
CEE3SPM—query and modify Language
Environment hardware condition enablement 189
CEE3SRP—set resume point 197
CECEDCOD—decompose a condition token 249
CEEGPID—retrieve the Language Environment
version and platform ID 304
CEEGQDT—retrieve q_data_token 309
CEETHDLR—register user condition handler 325
CEETHDLU—unregister user condition handler 335
CEEITOK—return initial condition token 348
CEEMRCE—move resume cursor explicit 376
CEEMRCR—move resume cursor relative to
handle cursor 382
CEENCOD—construct a condition token 400
CEESGL—signal a condition 449
Quick Reference Tables 97, 98
CICS, under 20
depth of conditions allowed 24
dumps, handling of conditions that arise when
processing 127
run-time options for
ABPERC 11
DEPTHCONDLMT 24
ERRCOUNT 28
TERMTHDACT 72
TRAP 86
XUFLOW 93
signaling condition with CEESGL 449
condition handling (continued)
unhandled condition
level of information produced for (TERMTHDACT run-time option) 72
user-written condition handler
allowing nested conditions in (DEPTHCONDLMT run-time option) 24
moving the resume cursor from a
(CEEMRCR) 382
registering (CEEHDLR) 325
retrieving q_data token (CEEGQDT) 309
TRAP run-time option and 86
unregistering (CEEHDLU) 336
vector facility and 90
condition information block
dumping information related to 130
returning initial condition token for (CEEITOK) 348
examples of 350, 354
Condition Information Block (CIB) 115
condition token
altering (CEEDCOD) 249, 251
constructing (CEENCOD) 400
examples of 403, 406
decomposing (CEEDCOD) 249
dumping information associated with (CEE3DMP) 130
input to CEESGL, using as 450
message corresponding to, getting
get, format, and dispatch message (CEEMSG) 394, 399
get, format, and store message in a buffer (CEEMGET) 364, 371
q_data_token from the ISI, using to retrieve (CEEGQDT) 309
examples of 312, 319
returning initial condition token (CEEITOK) 348
examples of 350, 354
SCEESAMP files for 105
condition_ID portion of condition token 402
conjgugate of complex math service (CEESxJCJG) 480, 481
constructing a condition token (CEENCOD) 400
control portion of condition token 250, 401
CONTROLLED storage for PL/I 33
convert character format to Lilian date (CEEDAYS) 242
examples of 242
convert Lilian date to character format (CEEDATE) 227
examples of 230, 234
COPY statement, in COBOL 105
cosine math service (CEESxCOS) 481
cotangent math service (CEESxCOTN) 485
COUNTRY run-time option
defaults of country_code parameter 515
relationship to setlocale() 23
syntax 22
country setting
default, querying with CEE3CTY callable service 120
default, setting with CEE3CTY callable service 120
country_code
country codes defaults table 515
in CEE3MCS callable service 172
examples of 173, 175
in CEE3MDS callable service 175
examples of 177, 179
in CEE3MTS callable service 179
examples of 181, 183
in CEEFMIDA callable service 267
examples of 268, 270
in CEEFMDT callable service 270
examples of 272, 274
in CEEFMTM callable service 280
examples of 281, 283
nonexistent country_code, specifying a 23
parameter 515
popping country_code off stack with CEE3CTY 121
pushing prior country_code to top of stack with CEE3CTY 121
querying (CEE3CTY) 120
examples of 123, 126
setting
with CEE3CTY callable service 120, 126
with COUNTRY run-time option 22
currency symbol
currency symbol defaults for a given country 515
default, obtaining with CEE3MCS callable service 172
examples of 173, 175
setting defaults for
with CEE3CTY callable service 120
with COUNTRY run-time option 22, 120

D

data types
definitions of Language Environment and HLL data types 110
data, external
relationship to ALL31 run-time option 15
DATAFIELD built-in function, in CEE3DMP 131
date and time
format
converting from character format to COBOL Lilian format (CEECBLDY) 203, 242
converting from character format to Lilian format (CEEDAYS) 242
converting from integers to seconds (CEESECI) 430
converting from Lilian format to character format (CEEDATE) 227
converting from seconds to character timestamp (CEEDATM) 234
converting from seconds to integers (CEESECS) 435
default formats for given country 515
setting default country (CEESXCTN) 485
setting default country (CEE3CTY) 120
Date and time (continued)

Getting date and time (CEELOCT) 360

Services

CEEDATE—convert Lilian date to character format 227
CEEDATM—convert seconds to character timestamp 234
CEEDAYS—convert date to Lilian format 203, 242
CEEDYWK—calculate day of week from Lilian date 261
CEEGGMT—get current Greenwich Mean Time 296
CEEGMTO—get offset from Greenwich Mean Time to local time 300
CEEISEC—convert integers to seconds 341
CEELOCT—get current local time 360
CEEQCEN—query the century window 407, 408
CEESCEN—set the century window 421
CEESECI—convert seconds to integers 430
CEESECS—convert timestamp to number of seconds 435

Quick Reference Tables 98

day of week, calculating with CEEDYWK 261

db2

POSIX run-time option and 56
DEBUG run-time option

syntax 24

Debug Tool

CEETEST callable service, invoking with 462
giving control to with TEST run-time option 77
invoking with CEETEST callable service 462
TEST run-time option and 77

debugging

COBOL batch 24
decimal separator

default, obtaining with CEE3MDS callable service 175
for countries, defaults 515
setting defaults for with CEE3CTY callable service 22, 120
table of defaults for a given country_code 515
DECIMAL-OVERFLOW condition 190, 191
DEPTHCONDLMNT run-time option

syntax 25
disability 529
documents, licensed xix
dump

CEE3ABD callable service and
abend with clean-up can generate Language Environment dump 114
abend without clean-up generates only system dump 113
Language Environment dump, requesting
CEE3DUMP default dump file 127, 130
examples of 133, 135
syntax description 127, 131
TEST compile-time option and 129
language written in 127
TERMTHDACT run-time option and 72

dynamic storage

allocating
additional heap, setting size of 215
initial heap, setting size of 33, 44
callable services for
CEE3RPH—set report heading 186, 189
CEECRHP—create new additional heap 215, 220
CEECZST—reallocate heap storage 223
CEEDSHP—discard heap 256, 260
CEEFRST—free heap storage 283, 288
CEEGTST—get heap storage 320, 325
Quick Reference Tables 99

initializing (STORAGE run-time option) 69

E

enablement
condition handling step
TRAP run-time option and 86

enabling exceptions
CEE3SPM callable service and 189, 195
CEESGL callable service and 449
XUFLOW run-time option and 93

enclave
return code of
obtaining current value of 135, 150, 156
setting new value of 150, 156, 196
termination with abend
using CEE3ABD for 113
enclave return code 135, 150, 156

ENV run-time option

syntax 26

ENVAR run-time option

overriding 27
POSIX run-time option and 27
syntax 27
equal (=)
in ENVAR run-time option 26

ERRCOUNT run-time option

CEESGL and 450
syntax 28

error function compliment math service
(CEESxERC) 488

error function math service (CEESxERF) 488

ERRUNIT run-time option

syntax 29

ESPIE 86

TRAP run-time option and 86

ESTAE

TRAP run-time option and 86

examples
CEE3ABD—terminate enclave with an abend 114, 115
CEE3CTY—set default country 123, 126
CEE3DMP—generate dump 133, 135
CEE3GRC—get enclave return code 137, 147
CEE3GRN—get name of routine that incurred condition 150, 156
CEE3LNG—set national language 167, 171
CEE3MCS—get default currency symbol 173, 175
examples (continued)

CEE3MDT—get default decimal separator 177, 179
CEE3MTS—get default thousands separator 181, 183
CEE3PRM—query parameter string 184, 186
CEE3RPH—set report heading 187, 189
CEE3SPM—query and modify Language Environment hardware condition enablement 193, 195
CEE3SRC—set enclave return code 197
CEE3USR—set or query user area fields 201, 203
CEECMI—store and load message insert data 211, 214
CEECRHP—create new additional heap 218, 220
CEECZST—reallocate (change size of) storage 223, 226
CEEDATE—convert Lilian date to character format 230, 234
CEEDATM—convert seconds to character format 237, 242
CEEDAYS—convert date to Lilian format 249
CEEDCOD—decompose a condition token 252, 255
CEEDSHP—discard heap 258, 260
CEEDYWK—calculate day of week from Lilian date 263, 266
CEEFMDA—get default date format 268, 270
CEEFMDT—get default date and time format 272, 274
CEEFMON—format monetary string 277
CEEFMTM—get default time format 281, 283
CEEFST—free heap storage with CEEGTST 286, 288
CEEFSTDS—get time into character string 293
CEEGMT—get current GMT 298, 300
CEEGMTO—get offset from GMT 304
CEEGPID—retrieve Language Environment version/platform ID 306, 309
CEEGQDT—get q_data_token 312, 319
CEEGTST—get heap storage 323, 325
CEEHDL—register user-written condition handler by itself 325
with CEE3GRD and CEE3SRC 137, 147
CEEHDLU—unregister user-written condition handler with CEE3DLR 337, 340
CEEISEC—convert integers to seconds 344, 347
CEEITOK—return initial condition token 350, 354
CEELCNV—query locale numeric conventions 358
CEELST—get local time 362
CEEMGET—get a message 368, 371
CEEMOUT—dispatch a message with CEEMGET 373
CEEMR—move resume cursor with CEEHDLR and CEEHDLU 388, 393
CEEMSG—get, format, and dispatch a message 396, 399
CEENCOD—construct a condition token 406
CEECEN—query century window 408, 410
CEEQDTC—query locale, date, and time conventions 413
examples (continued)

CEEQRYL—query active locale environment 418
CEESCEN—set century window 423, 426
CEESCOL—compare string collation weight 428
CEESDLG1—calculate logarithm base 10 510
CEESCE1—convert seconds to integers 432, 435
CEESECS—convert timestamp to number of seconds 439, 443
CEESETL—set locale operating environment 446
CEESGL—signal a condition 452, 455
CEESIMOD—perform modular arithmetic 474
CEESSAT2—calculate arctangent of 2 arguments 511
CEESSLOG—calculate logarithm base e 475
CEESTXF—transform string characters into collation weights 457
CEETEST—invoke debug tool 464, 466
declares in COBOL and PL/I for Language Environment data types 110
math services 474
exceptions
S/370 interrupt codes 192
EXEC CICS command
HANDLE ABEND
CBLPSPHPOR run-time option and 20
HANDLE AID
CBLPSPHPOR run-time option and 20
HANDLE CONDITION
CBLPSPHPOR run-time option and 20
POP HANDLE
CBLPSPHPOR run-time option and 20
PUSH HANDLE
CBLPSPHPOR run-time option and 20
EXECOPS run-time option
syntax 29
exponent underflow 93, 190, 191
exponential base e math service (CEESxEXP) 489
exponentiation math service (CEESxXPx) 507
external data
relationship to ALL31 run-time option 15

F
facility ID
CEENCOD callable service and 401
IGZ severity 0 and 1 conditions and the ERRCOUNT run-time option 28
feedback code
FEEDBACK data type 110
in callable services 109
omitting 109
FETCH statement for PL/I fetchable main
CEE3DMP ENCLAVE considerations 131
FILEDEF command for CMS
SYSABEND PRINTER 114
SYSDUMP PRINTER 114
FILEHiST run-time option
syntax 30
FILETAG run-time option 31
fixed-overflow condition, in CEE3SPM 190, 191
floating complex divide math service (CEESxDVD) 487
floating complex multiply math service (CEESxMLT) 497
FLOW run-time option
  syntax 33

G
gamma function math service (CEESxGMA) 491
general callable services
  CEE3GRC—get enclave return code 135, 147
  CEE3PRM—query parameter string 183, 186
  CEE3SRC—set enclave return code 196, 197
  CEE3USR—set or query user area fields 198, 203
  CEERAN0—calculate uniform random numbers 418, 421
  CEETEST—invoke debug tool 462, 467
  Quick Reference Tables 99
GOBACK statement
  RTEREUS run-time option and 64
Greenwich Mean Time (GMT)
  as seed parameter of CEERAN0 callable service 418
  getting offset to local time from (CEEGMTO) 300
    examples of 303, 304
  return Lilian date and Lilian seconds (CEEGMTO) 296
    examples of 298, 300
  Gregorian character string
    returning local time as a (CEELOCT) 361
    examples of 362

H
HANDLE ABEND EXEC CICS command
  CBLPSHPOP run-time option and 20
handle cursor
  moving resume cursor relative to (CEEEMCRCR) 382
    examples of 388
header files
  teawi.h (C)
    callable service declarations and 104
    condition token structure and 402
HEAP run-time option
  ALL31 run-time option and 14
  CEECRHP and 216
  different treatment under CICS 35
  STORAGE run-time option and 69
  syntax 33, 44
heap storage (continued)
callable services for
  CEE3RPH—set report heading 186
  CEECRHP—create new additional heap 215
  CEECZST—reallocate heap storage 221
 CEEDSHP—discard heap 256
  CEEFRST—free heap storage 283
  CEEGTST—get heap storage 320
  Quick Reference Tables 99
clearing after freeing, in STORAGE run-time option 69
discarding an entire heap (CEEDSHP) 256
  examples of 258
freeing (CEEFRST) 283
getting (CEEQTST) 320
heap element, changing size of (CEECZST) 221
heap ID
  heap ID 0 invalid in CEEDSHP 256
  returned by CEECRHP 215
  using to indicate which heap to discard, in
    CEEDSHP 256
  using to receive storage from a given heap, in
    CEEGTST 320
heap increment
  determining size of, with HEAP run-time option 34
HEAP run-time option and 33, 44
initial heap segment
  determining size of (HEAP run-time option) 34
initializing (STORAGE run-time option) 69
managing allocation of (HEAP run-time option) 33, 44
types of variables stored in 33, 44
HEAP64 run-time option 35
HEAPPOOLS64 run-time option 40
Heisei era 520
hyperbolic
  arctangent (CEESxATH) 478
  cosine (CEESxCOSH) 483
  sine (CEESxSNH) 503
  tangent (CEESxTNH) 507
hyperbolic arctangent math service (CEESxATH) 478
hyperbolic cosine math service (CEESxCOSH) 483
hyperbolic sine math service (CEESxSNH) 503
hyperbolic tangent math service (CEESxTNH) 507

I
I/O
  BELOWHEAP run-time option and 19
IGZ Facility ID
  ERRCOUNT run-time option and 28
imaginary part of complex math service (CEESxIMG) 491, 492
IMS (Information Management System) PLIST run-time option and 55
POSIX run-time option and 56, 84
  specifying a C application is running under 26
  include statement, in C and PL/I 105
INFOMSGFILTER run-time option 41
INFOMSGFILTER run-time option  *(continued)*
INFOMSGFILTER—eliminates unwanted informational messages  41
initial heap
allocating storage from (HEAP run-time option, CEEGTST)  33, 44, 321
restrictions against discarding  257
initial heap segment
determining size of (HEAP run-time option)  33, 44
reallocating (changing size of)  220
initializing storage
using options of CEECRHP callable service  216
using STORAGE run-time option  69, 71
input/output
See I/O
INQPCOPN run-time option
syntax  42
insert data
Language Environment-generated  401
user-created
cannot use CEEMOUT callable service to create  372
storing and loading  208
INSPRPREF preference file  79
instance specific information (ISI)
CEEDCOD callable service and  250
creating, when building a condition token  401
insert data is part of  401
maintaining a number of  50
overwriting  209
retrieving q_data_token from (CEEGQDT)  309
examples of  312, 319
storing address of message insert data in  208
using CEESGL callable service to create  449, 450
examples of  452, 455
integers
converting Lilian seconds to (CEESECI)  430
examples of  435
converting to Lilian seconds (CEESEC)  342
examples of  344, 347
INTERRUPT run-time option
debug tool and  43
syntax  43
TEST run-time option and  43
intrinsic functions, compatibility with CEELOCT callable service  361
invoking
Debug Tool
with CEETEST callable service  462, 467
with TEST run-time option  77
IOHEAP64 run-time option  44

L
language-specific condition handlers, use with XUFLOW run-time option  91
leawi file, description of  105
library
stack storage
specifying size of (LIBSTACK run-time option)  46
LIBSTACK run-time option
syntax  46
licensed documents  xix
Lilian date
calculate day of week from (CEEDYWK)  261
convert date to (CEEDAYS)  203, 242
convert output_seconds to (CEEISEC)  342
convert to character format (CEEDATE)  227
get current local date or time as a (CEELOCT)  360
get GMT as a (CEEGMT)  296
using as input to CEESECI callable service  431
local time
getting (CEELOCT)  360
locale services
CEEFMON—format monetary string  274
CEEFTDS—format date and time into character string  289
CEECNV—query locale numeric conventions  354
CEEQDT—return locale date and time as  410
CEEQRYL—query active locale environment  416
CEESCOL—compare string collation weight  426
CEESTL—set locale operating environment  443
CEESTXF—transform string character into collation weight  455
log gamma math service (CEESxLGM)  493
logarithm base 10 math service (CEESxLG1)  494
examples using  510
logarithm base 2 math service (CEESxLG2)  495
logarithm base e math service (CEESxLOG)  496
examples using  510
logarithm routines
base 10 (CEESxLG1)  494
base 2 (CEESxLG2)  495
base e (CEESxLOG)  496
log gamma (CEESxLGM)  493
LookAt message retrieval tool  xx

M
math services
absolute value (CEESxABS)  475
arccosine (CEESxACS)  476
arcsine (CEESxASN)  477
arctangent (CEESxATN)  479
arctangent2 (CEESxAT2)  480
conjugate of complex (CEESxCJG)  480
cosine (CEESxCOS)  481
cotangent (CEESxCOTN)  484
error function (CEESxERF)  488
error function compliment (CEESxERC)  487
exponential base e (CEESxEXP)  489
exponentiation (CEESxXPx)  507
floating complex divide (CEESxDVD)  487

J
Japanese
eras  520

K
keyboard  529
math services (continued)
floating complex multiply (CEEsXMLT)  497
gamma function (CEEsXGMA)  490
hyperbolic arctangent (CEEsXATH)  478
hyperbolic cosine (CEEsXCSH)  483
hyperbolic sine (CEEsXSNH)  503
hyperbolic tangent (CEEsXTNH)  507
imaginary part of complex (CEEsXIMG)  491
log gamma (CEEsXLGM)  493
logarithm base 10 (CEEsXLG1)  494
logarithm base 2 (CEEsXLG2)  495
logarithm base e (CEEsXLOG)  496
modular arithmetic (CEEsXMOD)  497
nearest integer (CEEsxNIN)  499
nearest whole number (CEEsxNWN)  500
positive difference (CEEsxDIM)  486
sine (CEEsxSIN)  501
square root (CEEsxSGT)  504
tangent (CEEsxTAN)  505
transfer of sign (CEEsxSGN)  500
truncation (CEEsxINT)  492
Meiji era  520
message
get, format, and dispatch a message (CEEMSG)  394, 396, 399
get, format, and store message in a buffer(CEEMGET)  364, 371
obtaining  364, 394
truncated  365
message file
pre-Language Environment options to Language Environment options  94
relationship to CBLQDA run-time option  21
message handling
nested conditions and  50
quick reference of callable services for  101
message retrieval tool, LookAt  xx
MinKow era  521
modular arithmetic math service (CEEsxMOD)  498
examples using  510
MSGFILE run-time option
different treatment under CICS  49
RPTOPTS options report and  47
RPTSTG storage report and  47
MSGQ run-time option
relationship to Instance Specific Information (ISIs)  50
syntax  50
MTF (Multitasking Facility)
allocating heap storage under (HEAP run-time option)  35
multiithreading
RPTSTG run-time option and  61
national language support (NLS)
default values for a specified country  515
quick reference of callable services for  99, 100, 101
specifying national language (NATLANG run-time option)  51
NATLANG run-time option
default of  51
syntax  51
natural log math service (CEEsxLOG)  496
nearest integer math service (CEEsxNIN)  499
nearest whole number math service (CEEsxNWN)  500
nested condition
getting name of routine that incurred a condition (CEE3GRN)  148
limiting (DEPTHCONDLMT run-time option)  24
MSGQ run-time options and  50
Notices  531
O
OCSTATUS run-time option
syntax  52
offset of condition, getting (CEE3GRO)  156, 158, 198
omitted parameter
how C indicates omission  107
ONCHAR built-in function, in CEE3DMP  131
ONCOUNT built-in function, in CEE3DMP  131
ONFILE built-in function, in CEE3DMP  131
ONKEY built-in function, in CEE3DMP  131
ONSOURCE built-in function, in CEE3DMP  131
options report, generating (RPTOPTS run-time option)  60
osplist macro  184
OSRUN command for CMS
PLIST run-time option and  54
out-of-storage condition  69, 70
P
parameter
list  407
querying  183
list format
PLIST run-time option and  54
PC run-time option
syntax  53
perror() function  49
picture string
defaults  515
tables of valid  232, 234
PL/I
built-in functions
DATAFIELD  131
ONCHAR  131
ONCOUNT  131
ONFILE  131
ONKEY  131
ONSOURCE  131
callable services, invoking from  103, 113
CEE3DMP callable service considerations
built-in function information dumped  131

N
national language
querying (CEE3LNG)  163
setting (NATLANG run-time option, CEE3LNG)  51, 163
PL/I (continued)

CEE3DMP callable service considerations
((continued))
  traceback information 129
  variables 129

CEE3RPH callable service equivalent to
PLIXHD 187

CEEHDLR callable service not allowed with 327
CEEHDLU callable service not allowed with 327
CEESGL callable service restriction 449

ERRCOUNT run-time option consideration 29

examples
  CEE3ABD—terminate enclave with an
abend 115
  CEE3CTY—set default country 125
  CEE3DMP—generate dump 134
  CEE3GRC—get enclave return code 146
  CEE3GRN—get name of routine that incurred
condition 155
  CEE3LNG—set national language 169
  CEE3MCS—get default currency symbol 174
  CEE3MDS—get default decimal separator 178
  CEE3MTS—get default thousands separator 182
  CEE3PRM—query parameter string 185
  CEE3SPM—query and modify Language
Environment hardware condition
  enablement 194
  CEE3SRC—set enclave return code 146
  CEE3USR—set or query user area fields 202
  CEELCNV—query locale numeric
  conventions 359
PL/I (continued)

examples (continued)
  CEELOCT—get current local time 363
  CEEMGET—get a message 370
  CEEMOUT—dispatch a message 374
  CEEMSG—get, format, and dispatch a
message 398
  CEENCOD—construct a condition token 405
  CEEQCN—query century window 409
  CEEQDTC—return locale date and time 414
  CEEQRYL—query active locale environment 418
  CEESCN—set century window 425
  CEESCOL—compare string collation weight 429
  CEESCI—convert seconds to integers 434
  CEESCES—convert timestamp to number of
seconds 442
  CEESETL—set locale operating
  environment 447
  CEESGL—signal a condition 454
  CEESTXF—transform string characters into
collation weights 458
  CEESxLOG—calculate log base e 511
  CEESxMOD—perform modular arithmetic 511
fetchable main
  CEE3DMP ENCLAVE considerations 131
  information produced after return from ERROR or
  FINISH ON-unit 72
  INTERRUPT option and 43
  mapping pre-Language Environment run-time options
to Language Environment run-time options 94
  mapping SPIE to TRAP 86
  MSGFILE run-time option and SYSPRINT 49
  MTF (Multitasking Facility)
    allocating heap storage under (HEAP run-time
    option) 35
    controlling number of tasks (PLITASKCOUNT
    run-time option) 76
    omitting fc parameter 108
  ON-units
    ZERODIVIDE 146
    semantics require exponent underflow be
    signaled 94
    variables, where stored 33
    XUFLOW run-time option considerations 94
  PLIST run-time option
    CICS ignores this option 55
    syntax 55
  PLITASKCOUNT—control the maximum number of
  active tasks
    syntax 55
  POP function
    using to change the current country setting, in
    CEE3CTY 121
    using to change the current national language
    setting, in CEE3LNG 164, 165
    using to query or modify the enablement of hardware
    conditions, in CEE3SPM 191
  positive difference math service (CEESxDIM) 486
  POSIX run-time option
    ANSI C routines and 56, 85
    service routine vector in API interface and 56, 85
POSIX run-time option (continued)
syntax 85
previously allocated storage, changing size of
(CEECZST) 221
equivalence of 223, 226
program interrupts
CEE3SPM and 189
math services and 492, 498
table of 5/370 interrupt codes 192
TRAP run-time option and 86
XUFLOW run-time option and 93
program mask 190
PRTUNIT run-time option
syntax 57
PUNUNIT run-time option
syntax 58
PUSH function
using to change the current country setting, in
CEE3CTY 121
using to change the current national language, in
CEE3LNG 164
using to query or modify the enablement of hardware
conditions, in CEE3SPM 191
Q
q_data_token, in CEESGL
creating 450
retrieving from the ISI (CEEGQDT) 309
QUERY function
using to check the current country setting, in
CEE3CTY 121
using to check the current national language, in
CEE3LNG 164
using to check the enablement of hardware
conditions, in CEE3SPM 190
Quick Reference Tables 3
R
random numbers
generation of (CEERAN0) 418
RDRUNIT run-time option
syntax 58
reason code
CEE3DMP callable service and 131
dumps and 131
RECPAD run-time option
syntax 59
REDIR run-time option
syntax description 59
redirections
of stderr, stdout and stdin output 49, 59
REDIR run-time option and 59
register
save area, as component of dsa_alloc_value 70
report
generating options report (RPTOPTS run-time
option) 60
resume
cursor
moving (CEEMRCR) 382
setting resume point (CEE3SRP) 197
reusability of an environment 63
routine that incurred condition, getting
(CEE3GRN) 148, 150, 156
RPTOPTS run-time option
relationship to MSGFILE run-time option 60
sample options report generated by 61
syntax 61
RPTSTG run-time option
CEE3TST and 321
storage report generated by
setting heading for 186
RTEREUS run-time option
CICS ignores this option 64
syntax 63
run-time options
ABPERC—percolate an abend 11
AIXBLD—invoke AMS for COBOL 13
ALL31—indicate whether application runs in
AMODE(31) 14
ANYHEAP—control unrestricted library heap
storage 16
ARGPARSE—specify whether arguments are
parsed 17
AUTOTASK—specify whether Fortran MTF is to be
used 18
BELOWHEAP—control library heap storage below
16M 19
CBLOPTS—specify format of COBOL argument 20
CBLPSHPOP—control CICS commands 20
CBLQDA—control COBOL QSAM 21
CHECK—detect checking errors 22
COUNTRY—specify default date/time formats 22
DEBUG—activate COBOL batch debugging 24
DEPTHCONDLMT—limit extent of nested
conditions 24
ENV—specify operating environment for C
application 27
ERRCOUNT—specify number of errors allowed 28
ERRUNIT—specify unit number to which error
information is directed 29
EXECOPS—let run-time options be specified on
command line 29
FILEHIST—specify whether to allow a file definition
to be changed at run time 30
FILETAG—specify whether to allow AUTOTAG /
AUTOVT. 31
FLOW—control FLOW output for OS/VS
COBOL 33
HEAP—control allocation of heaps 33, 44
HEAP64—controls allocation of user heap
storage 35
HEAPPools64—controls optional user heap
storage management algorithm 40
INQPCOPN—control value in OPENED specifier of
INQUIRE by unit statement 42
INTERRUPT—cause attentions to be recognized by
Language Environment 43
run-time options (continued)

IOHEAP64 — controls allocation of I/O heap storage 44
LIBSTACK — control library stack storage 46
MSGQ — specify number of ISI blocks allocated 50
NATLANG — specify national language 51
OCSTATUS — control checking of file existence and whether file deletion occurs 52
PC — control whether Fortran status common blocks are shared among load modules 53
PLIST — specify format of C arguments 55
PLITASKCOUNT — control the maximum number of active tasks 55
PRTUNIT — specifies unit number used for PRINT and WRITE statements 57
PUNUNIT — specifies unit number used for PUNCH statements 58
Quick Reference Tables 3, 5, 6
RDRUNIT — specifies unit number used for READ statements 58
RECPAD — specifies whether a formatted input record is padded with blanks 59
REDIR — specify redirections for C output 59
report of options specified
  generating heading for 186
  language report is written in 51, 60, 61
  not generated if application abends 61
  sample of 61
RPTOPTS — generate a report of run-time options used 60
RTEREUS — initialize a reusable COBOL environment 63
SIMVRD — specify VSAM KSDS for COBOL 64
STACK — allocate stack storage 65, 68
STORAGE — control storage 69
TERMTHDACT — specify type of information generated with unhandled error 71
TEST — indicate debug tool to gain control 77
TRACE — activate Language Environment run-time library tracing 84
TRAP — handle abends and program interrupts 86
UPS1 — set UPS1 switches 88
USRHDLR — register a user condition handler at stack frame 0 89
VCTRSAVE — use vector facility 90
XUFLOW — specify program interrupt due to exponent underflow 93

S

safe condition 450
SCEESAMP sample library declaration files in 105
  sending product messages to a file (CEEMSG) 394
    examples of 396, 399
  sending user-defined message string to file (CEEMOUT) 371
    examples of 373, 375
SET function (continued)
  changing the enablement of hardware conditions with 190
  changing the national language with 164
  severity of a condition
    CEESGL and 450
    ERRCOUNT run-time option and 28
    severity levels that cause the debug tool to gain control 77
    TERMTHDACT run-time option and 72
  shortcut keys 529
Showa era 520
SIGNIFICANCE condition 190, 191
SIMVRD run-time option syntax 64
sine math service (CEESxSIN) 502
slash (/) NOEXECOPS alters behavior of 30
  specifying in parameters of TEST run-time option 78
SPIE run-time option 86
square root math service (CEESxSQT) 505
SSRANGE option for VS COBOL II 22
stack
  frame
    CEE3DMP callable service and 130
    library, allocating (LIBSTACK run-time option) 46
    storage
      ALL31 run-time option and 14
      allocating (STACK run-time option) 65, 68
      initializing (STACK run-time option) 69
      RPTSTG run-time option and 62
      setting initial stack segment (STACK run-time option) 65, 68
      threads and 62
    user, allocating (STACK run-time option) 65, 68
STACK run-time option syntax 65, 68
  using with RPTSTG to tune the stack 62
STAE
  C options 86
  VS COBOL II options 86
stderr
  MSGFILE run-time option and 49
  REDIR run-time option and 59
  redirecting output from 59
stdin 59
  MSGFILE run-time option and 59
  REDIR run-time option and 59
  redirecting output from 59
stdout 59
  MSGFILE run-time option and 59
  REDIR run-time option and 59
  redirecting output from 59
storage
  additional heaps, creating new (CEECRHP) 215
  discarding an entire heap (CEEDSHP) 256
  freeing
    clearing after freeing (STORAGE run-time option) 69
storage (continued)
freeing (continued)
  discarding an entire heap (CEEDSHP) 256, 260
  freeing additional heap (CEEFRST) 283, 288
getting
  heap storage (CEEGTST) 320
initializing (STORAGE run-time option) 69
options for
  ANYHEAP—control library heap 16
  BELOWHEAP—control library heap below 16M 19
  HEAP—control initial heap 33, 44
  LIBSTACK—control library stack storage 46
  STACK—control thread stack storage 65, 68
  STORAGE—control initial heap or stack 69
report
  language written in 51, 60, 61
  not generated if application abends 62
  setting heading for 186
  STACK run-time option and 62
services
  CEE3RPH—set report heading 186
  CEECRHP—create new additional heap 215
  CEECZST—reallocate (change size of) storage 221
  CEDSHP—discard heap 256
  CEEFRST—free heap storage 284
  CEEGTST—get heap storage 320
thread-level heap, controlling (THREADHEAP run-time option) 79
  tuning
    additional heap, setting size of (CEECRHP) 215
    anywhere heap, setting size of (ANYHEAP) 16
    below heap, setting size of (BELOWHEAP) 19
    initial heap, setting size of (HEAP) 33, 44
    initial stack segment, setting size of (STACK) 65, 68
    library stack storage, setting size of (LIBSTACK) 46
    with CEEGTST 320
    with STORAGE run-time option 69
STORAGE run-time option
  CEECZST callable service and 221
  CEEFRST callable service and 284
  syntax 69
straight double quote ("
  initializing storage with 69, 71
  specifying in parameters of TEST run-time option 78, 79
straight single quote ('
  initializing storage with 69, 71
  specifying in parameters of TEST run-time option 78, 79
symbol
  table, generated by CEE3DMP 129
syntax diagrams
  how to read 6
SYSABEND PRINTER 114
SYSDUMP PRINTER 114
  default destinations of MSGFILE run-time option 48
SYSOUT
  T
  Taisho era  520
  Taiwan era  521
tangent math service (CEESxTAN) 506
task
  controlling number of tasks under MTF (PLITASKCOUNT option) 76
termination
  enclave
    CEE3ABD and 113
termination imminent step
  TERMTHDACT run-time option and 76
  TERMTTHDACT run-time option
    CEE3DMP and 76
    CESE transient data queue and 76
different treatment under CICS 76
  syntax 72
TEST compile-time option 129
TEST run-time option
  syntax 77
thousands separator
  obtaining default of (CEE3CTY) 120
  setting defaults for (COUNTRY) 22
thread
  multiple
    storage report for (RPTSTG run-time option) 62
  THREADHEAP—control the allocation of thread-level heap storage
    syntax 79
time, getting local (CEEOCT) 360
timestamp 234, 435
TRACE run-time option
  syntax 84
trace, generating a 72
transfer of sign math service (CEESxSGN) 501
TRAP run-time option
  ABPERC run-time option and 11
  syntax 86
truncation math service (CEESxINT) 493
U
UNDERFLOW condition 93, 190, 191
UPS1 run-time option
  syntax 88
USE FOR DEBUGGING declarative 24
user
  area fields 196, 198, 200
  heap (initial heap)
    allocating storage from 33, 44, 321
    restrictions against discarding 257
  stack, controlling (STACK run-time option) 62
user-written condition handler
  CEE3SPM callable service and 190
  CEEMRCR callable service and 382
  registering with CEEHDLR callable service 325
  unregistering 336
  VCTRSAVE run-time option and 90
USRHDLR run-time option
  syntax 89
V
valid condition  449
VCTRSAVE run-time option
   syntax  90
vector facility  90
VSAM
   KSDS  13, 64
   RRDS  13

W
WITH DEBUGGING MODE clause for COBOL  24

X
XPLINK run-time option  91
   XPLink  91
XUFLOW run-time option
   syntax  93
# Readers’ Comments — We’d Like to Hear from You

z/OS
Language Environment
Programming Reference

Publication No. SA22-7562-06

**Overall, how satisfied are you with the information in this book?**

<table>
<thead>
<tr>
<th>Overall satisfaction</th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**How satisfied are you that the information in this book is:**

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accurate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to find</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to understand</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Well organized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Please tell us how we can improve this book:**

---

Thank you for your responses. May we contact you?  
☐ Yes  ☐ No

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company or Organization</th>
<th>Phone No.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Readers’ Comments — We’d Like to Hear from You

SA22-7562-06

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

Postage will be paid by addressee.

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

Fold and Tape
Please do not staple
Fold and Tape

SA22-7562-06
Program Number: 5694-A01, 5655-G52

Printed in USA