Contents

Chapter About This Manual
Related Publications .......................................................... xxiii
Notational Conventions ...................................................... xxiv

Chapter 1 Intrinsic Procedures
Overview of Intrinsic Procedures ........................................... 1-1
Availability of Intrinsic Procedures ........................................ 1-2
Intrinsic Subroutines and Functions ..................................... 1-2
  Intrinsic Subroutines ..................................................... 1-3
  Elemental and Nonelemental Subroutines ......................... 1-3
Intrinsic Functions .......................................................... 1-3
  Generic and Specific Intrinsic Function Names .................. 1-4
  Elemental Functions ..................................................... 1-5
  Inquiry Functions ........................................................ 1-5
  Transformational Functions ............................................. 1-6
INTRINSIC Attribute and Statement ..................................... 1-7
  Documenting Intrinsic Procedures .................................. 1-7
  Intrinsic Procedures as Actual Arguments ...................... 1-7
Nonstandard Intrinsic Procedures ...................................... 1-9
Data Representation Models ............................................. 1-10
  Data Representation Model Intrinsics .............................. 1-10
  The Bit Model ............................................................ 1-11
  The Integer Number System Model ............................... 1-11
The Real Number System Model ........................................ 1-12
Functional Categories of Intrinsic Procedures ............... 1-13
Generic and Specific Intrinsic Summary ....................... 1-14
  Summary of Generic and Specific Intrinsic Names ...... 1-15
Intrinsic Procedure Specifications ................................ 1-139
  ABS(A) .................................................................... 1-139
  ACHAR(I) .................................................................. 1-140
  ACOS(X) ................................................................... 1-141
  ACOSD(X) ................................................................. 1-142
  ACOSH(X) ................................................................. 1-143
  ADJUSTL(STRING) .................................................. 1-144
  ADJUSTR(STRING) ................................................... 1-145
  AIMAG(Z) ................................................................ 1-146
  AINT(A, KIND) ......................................................... 1-147
  ALL(MASK, DIM) ....................................................... 1-148
  ALLOCATED(ARRAY) ................................................ 1-149
  AND(I, J) .................................................................. 1-150
  ANINT(A, KIND) ....................................................... 1-151
  ANY(MASK, DIM) ....................................................... 1-152
  ASIN(X) .................................................................... 1-153
  ASIND(X) ................................................................. 1-154
  ASINH(X) .................................................................. 1-155
  ASSOCIATED(POINTER, TARGET) ....................... 1-156
  ATAN(X) .................................................................... 1-157
  ATAN2(Y, X) .............................................................. 1-158
  ATAN2D(Y, X) ......................................................... 1-159
  ATAND(X) .................................................................. 1-160
  ATANH(X) .................................................................. 1-161
  BADDRESS(X) .......................................................... 1-162
  BIT_SIZE(I) ............................................................... 1-163
  BTEST(I, POS) ......................................................... 1-164
  CEILING(A) ............................................................... 1-165
  CITATION(MONTH, DAY) ......................................... 1-166
  COALESCE(A, B) ......................................................... 1-167
  COS(X) ...................................................................... 1-168
  COSD(X) ................................................................... 1-169
  COSH(X) .................................................................... 1-170
  COTAN(X) .................................................................. 1-171
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(I, KIND)</td>
<td>1-170</td>
</tr>
<tr>
<td>CMPLX(X, Y, KIND)</td>
<td>1-171</td>
</tr>
<tr>
<td>CONJG(Z)</td>
<td>1-172</td>
</tr>
<tr>
<td>COS(X)</td>
<td>1-173</td>
</tr>
<tr>
<td>COSD(X)</td>
<td>1-174</td>
</tr>
<tr>
<td>COSH(X)</td>
<td>1-175</td>
</tr>
<tr>
<td>COUNT(MASK, DIM)</td>
<td>1-176</td>
</tr>
<tr>
<td>CPU_TIME(TIME)</td>
<td>1-178</td>
</tr>
<tr>
<td>CSHIFT(ARRAY, SHIFT, DIM)</td>
<td>1-179</td>
</tr>
<tr>
<td>DATE_AND_TIME(DATE, TIME, ZONE, VALUES)</td>
<td>1-181</td>
</tr>
<tr>
<td>DBLE(A)</td>
<td>1-183</td>
</tr>
<tr>
<td>DFLOAT(A)</td>
<td>1-184</td>
</tr>
<tr>
<td>DIGITS(X)</td>
<td>1-185</td>
</tr>
<tr>
<td>DIM(X, Y)</td>
<td>1-186</td>
</tr>
<tr>
<td>DNUM(I)</td>
<td>1-187</td>
</tr>
<tr>
<td>DOT_PRODUCT(VECTOR_A, VECTOR_B)</td>
<td>1-188</td>
</tr>
<tr>
<td>DPROD(X, Y)</td>
<td>1-189</td>
</tr>
<tr>
<td>DREAL(A)</td>
<td>1-190</td>
</tr>
<tr>
<td>DSIGN</td>
<td>1-191</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>1-192</td>
</tr>
<tr>
<td>EPSILON(X)</td>
<td>1-194</td>
</tr>
<tr>
<td>EXP(X)</td>
<td>1-195</td>
</tr>
<tr>
<td>EXPONENT(X)</td>
<td>1-196</td>
</tr>
<tr>
<td>FLOOR(A)</td>
<td>1-197</td>
</tr>
<tr>
<td>FRACTION(X)</td>
<td>1-198</td>
</tr>
<tr>
<td>FREE(A)</td>
<td>1-199</td>
</tr>
<tr>
<td>HFIX(A)</td>
<td>1-200</td>
</tr>
<tr>
<td>HUGE(X)</td>
<td>1-201</td>
</tr>
<tr>
<td>IABS(A)</td>
<td>1-202</td>
</tr>
<tr>
<td>IACHAR(C)</td>
<td>1-202</td>
</tr>
<tr>
<td>IADDR(X)</td>
<td>1-203</td>
</tr>
<tr>
<td>IAND(I, J)</td>
<td>1-204</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>IBCLR(I, POS)</td>
<td>1-205</td>
</tr>
<tr>
<td>IBITS(I, POS, LEN)</td>
<td>1-206</td>
</tr>
<tr>
<td>IBSET(I, POS)</td>
<td>1-207</td>
</tr>
<tr>
<td>ICHAR(C)</td>
<td>1-208</td>
</tr>
<tr>
<td>IDIM(X, Y)</td>
<td>1-209</td>
</tr>
<tr>
<td>IEOR(I, J)</td>
<td>1-210</td>
</tr>
<tr>
<td>IJINT(A)</td>
<td>1-211</td>
</tr>
<tr>
<td>IMAG(A)</td>
<td>1-212</td>
</tr>
<tr>
<td>INDEX(STRING, SUBSTRING, BACK)</td>
<td>1-213</td>
</tr>
<tr>
<td>INT(A, KIND)</td>
<td>1-214</td>
</tr>
<tr>
<td>INT1(A)</td>
<td>1-215</td>
</tr>
<tr>
<td>INT2(A)</td>
<td>1-216</td>
</tr>
<tr>
<td>INT4(A)</td>
<td>1-217</td>
</tr>
<tr>
<td>INT8(A)</td>
<td>1-217</td>
</tr>
<tr>
<td>INUM(I)</td>
<td>1-218</td>
</tr>
<tr>
<td>IOR(I, J)</td>
<td>1-219</td>
</tr>
<tr>
<td>IQINT(A)</td>
<td>1-220</td>
</tr>
<tr>
<td>ISHFT(I, SHIFT)</td>
<td>1-220</td>
</tr>
<tr>
<td>ISHFTC(I, SHIFT, SIZE)</td>
<td>1-221</td>
</tr>
<tr>
<td>ISIGN(A, B)</td>
<td>1-223</td>
</tr>
<tr>
<td>ISNAN(X)</td>
<td>1-224</td>
</tr>
<tr>
<td>IXOR(I, J)</td>
<td>1-224</td>
</tr>
<tr>
<td>JNUM(I)</td>
<td>1-226</td>
</tr>
<tr>
<td>KIND(X)</td>
<td>1-226</td>
</tr>
<tr>
<td>LBOUND(ARRAY, DIM)</td>
<td>1-227</td>
</tr>
<tr>
<td>LEN(STRING)</td>
<td>1-229</td>
</tr>
<tr>
<td>LEN_TRIM(STRING)</td>
<td>1-230</td>
</tr>
<tr>
<td>LGE(STRING_A, STRING_B)</td>
<td>1-231</td>
</tr>
<tr>
<td>LGT(STRING_A, STRING_B)</td>
<td>1-232</td>
</tr>
<tr>
<td>LLE(STRING_A, STRING_B)</td>
<td>1-233</td>
</tr>
<tr>
<td>LLT(STRING_A, STRING_B)</td>
<td>1-234</td>
</tr>
<tr>
<td>LOC(X)</td>
<td>1-235</td>
</tr>
</tbody>
</table>
LOG(X) ................................................................. 1-235
LOG10(X) ............................................................ 1-236
LOGICAL(L, KIND) .............................................. 1-237
LSHFT(I, SHIFT) ................................................. 1-238
LSHIFT(I, SHIFT) .............................................. 1-238
MALLOC(I) .......................................................... 1-239
MATMUL(MATRIX_A, MATRIX_B) ....................... 1-240
MAX(A1, A2, A3, ...) ........................................... 1-242
MAXEXPONENT(X) .............................................. 1-243
MAXLOC(ARRAY, MASK) ...................................... 1-244
MAXVAL(ARRAY, DIM, MASK) ......................... 1-246
MCLOCK() ......................................................... 1-248
MERGE(TSOURCE, FSOURCE, MASK) ................. 1-249
MIN(A1, A2, A3, ...) .......................................... 1-250
MINEXPONENT(X) ............................................. 1-251
MINLOC(ARRAY, MASK) ..................................... 1-252
MINVAL(ARRAY, DIM, MASK) ............................. 1-254
MOD(A, P) ........................................................ 1-256
MODULO(A, P) .................................................. 1-257
MVBITS(FROM, FROMPOS, LEN, TO, TOPOS) .... 1-258
NEAREST(X, S) .................................................. 1-259
NINT(A, KIND) .................................................. 1-260
NOT(I) .............................................................. 1-261
OR(I, J) .............................................................. 1-262
PACK(ARRAY, MASK, VECTOR) ......................... 1-263
PRECISION(X) .................................................. 1-265
PRESENT(A) ...................................................... 1-266
PRODUCT(ARRAY, DIM, MASK) ......................... 1-267
RADIX(X) .......................................................... 1-269
RANDOM_NUMBER(HARVEST) ............................ 1-270
RANDOM_SEED(SIZE, PUT, GET) ....................... 1-271
RANGE(X) ........................................................ 1-272
REAL(A, KIND) ............................................................. 1-273
REPEAT(STRING, NCOPIES) ........................................ 1-274
RESHAPE(SOURCE, SHAPE, PAD, ORDER) ............. 1-275
RNUM(I) ......................................................................... 1-277
RRSPACING(X) ............................................................ 1-277
RSHIFT(I, SHIFT) ......................................................... 1-278
RSHIFT(I, SHIFT) .......................................................... 1-279
SCALE(X, I) ............................................................... 1-279
SCAN(STRING, SET, BACK) ...................................... 1-280
SELECTED_INT_KIND(R) ............................................ 1-281
SELECTED_REAL_KIND(P, R) .................................... 1-282
SET_EXPONENT(X, I) ................................................. 1-284
SHAPE(SOURCE) ........................................................ 1-285
SIGN(A, B) .................................................................... 1-286
SIN(X) .......................................................................... 1-286
SIND(X) ......................................................................... 1-287
SINH(X) ......................................................................... 1-288
SIZE(ARRAY, DIM) ....................................................... 1-289
SPACING(X) ................................................................. 1-290
SPREAD(SOURCE, DIM, NCOPIES) ... 1-291
SQR(X) ........................................................................... 1-292
SUM(ARRAY, DIM, MASK) ........................................... 1-293
SYSTEM_CLOCK(COUNT, COUNT_RATE, COUNT_MAX) . 1-295
TAN(X) .......................................................................... 1-296
TAND(X) ........................................................................ 1-297
TANH(X) ........................................................................ 1-298
TINY(X) .......................................................................... 1-299
TRANSFER(SOURCE, MOLD, SIZE) .......................... 1-300
TRANSPOSE(MATRIX) ................................................ 1-302
TRIM(STRING) ............................................................. 1-303
UBOUND(ARRAY, DIM) ................................................ 1-304
Contents

UNPACK(VECTOR, MASK, FIELD) ........................................... 1-306
VERIFY(STRING, SET, BACK) ........................................ 1-308
XOR(I, J) ........................................................................ 1-310

Chapter 2 Portability Functions

ABORT .............................................................................. 2-2
ACCESS ........................................................................... 2-3
ALARM ........................................................................... 2-5
AMOD ............................................................................. 2-6
BEEPQQ ........................................................................ 2-7
BESJ0, BESJ1, BESJN, BESY0, BESY1, BESYN ........... 2-8
BIC, BIS ........................................................................ 2-10
BIT ................................................................................. 2-10
BSEARCHQQ ............................................................... 2-11
CDFLOAT ...................................................................... 2-13
CHANGEDIRQQ .......................................................... 2-14
CHANGEDRIVEQQ ..................................................... 2-15
CHDIR ........................................................................... 2-16
CHMOD ......................................................................... 2-17
CLEARSTATUSFPQQ.................................................. 2-19
CLOCK ........................................................................... 2-20
CLOCKX ....................................................................... 2-20
COMMITQQ ................................................................... 2-21
COMPL .......................................................................... 2-23
CSMG ............................................................................. 2-24
CTIME ............................................................................ 2-25
DATE .............................................................................. 2-26
DATE4 ........................................................................... 2-27
DBESJ0, DBESJ1, DBESJN, DBESY0, DBESY1, DBESYN ......................................... 2-28
DCLOCK ......................................................................... 2-30
DELDIRQQ ..................................................................... 2-31
DELFIELDSQQ ............................................................. 2-32
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETDRIVEDIRQQ</td>
<td>2-71</td>
</tr>
<tr>
<td>GETDRIVESIZEQQ</td>
<td>2-73</td>
</tr>
<tr>
<td>GETDRIVESQQ</td>
<td>2-75</td>
</tr>
<tr>
<td>GETENV</td>
<td>2-76</td>
</tr>
<tr>
<td>GETENVQQ</td>
<td>2-77</td>
</tr>
<tr>
<td>GETFILEINFOQQ</td>
<td>2-79</td>
</tr>
<tr>
<td>GETGID</td>
<td>2-83</td>
</tr>
<tr>
<td>GETLASTERROR</td>
<td>2-84</td>
</tr>
<tr>
<td>GETLASTERRORQQ</td>
<td>2-85</td>
</tr>
<tr>
<td>GETLOG</td>
<td>2-87</td>
</tr>
<tr>
<td>GETPID</td>
<td>2-88</td>
</tr>
<tr>
<td>GETPOS</td>
<td>2-89</td>
</tr>
<tr>
<td>GETSTATUSFPQQ</td>
<td>2-89</td>
</tr>
<tr>
<td>GETSTRQQ</td>
<td>2-92</td>
</tr>
<tr>
<td>GETTIM</td>
<td>2-93</td>
</tr>
<tr>
<td>GETTIMEOFDAY</td>
<td>2-94</td>
</tr>
<tr>
<td>GETUID</td>
<td>2-95</td>
</tr>
<tr>
<td>GMTIME</td>
<td>2-96</td>
</tr>
<tr>
<td>HOSTNAM</td>
<td>2-97</td>
</tr>
<tr>
<td>HOSTNM</td>
<td>2-98</td>
</tr>
<tr>
<td>IARG</td>
<td>2-99</td>
</tr>
<tr>
<td>IARGC</td>
<td>2-100</td>
</tr>
<tr>
<td>IDATE</td>
<td>2-101</td>
</tr>
<tr>
<td>IDATE4</td>
<td>2-102</td>
</tr>
<tr>
<td>IDFLOAT</td>
<td>2-103</td>
</tr>
<tr>
<td>IEEE_FLAGS</td>
<td>2-104</td>
</tr>
<tr>
<td>IEEE_HANDLER</td>
<td>2-108</td>
</tr>
<tr>
<td>IERRNO</td>
<td>2-109</td>
</tr>
<tr>
<td>IFL_RUNTIME_INIT</td>
<td>2-110</td>
</tr>
<tr>
<td>IFLOAT</td>
<td>2-111</td>
</tr>
<tr>
<td>IFLOATI</td>
<td>2-112</td>
</tr>
<tr>
<td>IFLOATJ</td>
<td>2-112</td>
</tr>
</tbody>
</table>
IMOD ................................................................. 2-113
INMAX ............................................................... 2-114
INTC ...................................................................... 2-114
IRAND ................................................................. 2-115
IRANDM .............................................................. 2-116
IRANGET ............................................................ 2-116
IRANSET ............................................................ 2-117
ISATTY .................................................................. 2-117
ITIME ..................................................................... 2-118
JABS ...................................................................... 2-119
JDATE ...................................................................... 2-119
JDATE4 ................................................................. 2-120
KILL ....................................................................... 2-121
LCWRQQ .............................................................. 2-122
LEADZ .................................................................... 2-123
LNBLNK .................................................................. 2-124
LONG ...................................................................... 2-125
LSTAT ..................................................................... 2-125
LTIME ................................................................. 2-126
MAKEDIRQQ .......................................................... 2-128
MATHERRQQ ......................................................... 2-129
NARGS .................................................................... 2-132
NUMARG ............................................................. 2-133
PACKTIMEQQ .......................................................... 2-134
PEEKCHARQQ .......................................................... 2-136
PERROR ............................................................... 2-137
POPCNT ................................................................. 2-138
POPPAR ................................................................. 2-138
PUTC ...................................................................... 2-139
QRANSET ............................................................. 2-140
QSORT ................................................................. 2-140
RAISEQQ ............................................................. 2-141
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAN</td>
<td>2-143</td>
</tr>
<tr>
<td>RAND</td>
<td>2-144</td>
</tr>
<tr>
<td>RANDOM</td>
<td>2-146</td>
</tr>
<tr>
<td>RANDU</td>
<td>2-147</td>
</tr>
<tr>
<td>RANF</td>
<td>2-148</td>
</tr>
<tr>
<td>RANGET</td>
<td>2-149</td>
</tr>
<tr>
<td>RANSET</td>
<td>2-149</td>
</tr>
<tr>
<td>RENAME</td>
<td>2-150</td>
</tr>
<tr>
<td>RENAMEFILEQQ</td>
<td>2-151</td>
</tr>
<tr>
<td>RINDEX</td>
<td>2-152</td>
</tr>
<tr>
<td>RTC</td>
<td>2-153</td>
</tr>
<tr>
<td>RUNQQ</td>
<td>2-154</td>
</tr>
<tr>
<td>SCWRQQ</td>
<td>2-155</td>
</tr>
<tr>
<td>SCANENV</td>
<td>2-156</td>
</tr>
<tr>
<td>SEED</td>
<td>2-157</td>
</tr>
<tr>
<td>SECNDS</td>
<td>2-157</td>
</tr>
<tr>
<td>SETCONTROLFPPQQ</td>
<td>2-159</td>
</tr>
<tr>
<td>SETDAT</td>
<td>2-161</td>
</tr>
<tr>
<td>SETENVQQ</td>
<td>2-162</td>
</tr>
<tr>
<td>SETERRORMODEQQ</td>
<td>2-164</td>
</tr>
<tr>
<td>SETFILEACCESSQQ</td>
<td>2-166</td>
</tr>
<tr>
<td>SETFILETIMEQQ</td>
<td>2-167</td>
</tr>
<tr>
<td>SETTIM</td>
<td>2-169</td>
</tr>
<tr>
<td>SHIFTL</td>
<td>2-170</td>
</tr>
<tr>
<td>SHIFTR</td>
<td>2-171</td>
</tr>
<tr>
<td>SHORT</td>
<td>2-172</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>2-173</td>
</tr>
<tr>
<td>SIGNALQQ</td>
<td>2-176</td>
</tr>
<tr>
<td>SLEEP</td>
<td>2-179</td>
</tr>
<tr>
<td>SLEEPQQ</td>
<td>2-179</td>
</tr>
<tr>
<td>SORTQQ</td>
<td>2-180</td>
</tr>
<tr>
<td>SPLITPATHQQ</td>
<td>2-182</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
</tr>
<tr>
<td>SRAND</td>
<td>2-184</td>
</tr>
<tr>
<td>SSWRQQ</td>
<td>2-185</td>
</tr>
<tr>
<td>STAT</td>
<td>2-186</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>2-188</td>
</tr>
<tr>
<td>SYSTEMQQ</td>
<td>2-188</td>
</tr>
<tr>
<td>TIME</td>
<td>2-190</td>
</tr>
<tr>
<td>TIMEF</td>
<td>2-191</td>
</tr>
<tr>
<td>TOPEN</td>
<td>2-192</td>
</tr>
<tr>
<td>TCLOSE</td>
<td>2-193</td>
</tr>
<tr>
<td>TREAD</td>
<td>2-194</td>
</tr>
<tr>
<td>TTYNAM</td>
<td>2-195</td>
</tr>
<tr>
<td>TWRITE</td>
<td>2-196</td>
</tr>
<tr>
<td>UNLINK</td>
<td>2-197</td>
</tr>
<tr>
<td>UNPACKTIMEQQ</td>
<td>2-197</td>
</tr>
<tr>
<td>National Language Support Routines</td>
<td>2-199</td>
</tr>
<tr>
<td>NLSEnumCodepages</td>
<td>2-204</td>
</tr>
<tr>
<td>NLSEnumLocales</td>
<td>2-205</td>
</tr>
<tr>
<td>NLSGetEnvironmentCodepage</td>
<td>2-206</td>
</tr>
<tr>
<td>NLSGetLocale</td>
<td>2-207</td>
</tr>
<tr>
<td>NLSGetLocaleInfo</td>
<td>2-208</td>
</tr>
<tr>
<td>NLSSetEnvironmentCodepage</td>
<td>2-219</td>
</tr>
<tr>
<td>NLSSetLocale</td>
<td>2-221</td>
</tr>
<tr>
<td>Locale Formatting Procedures</td>
<td>2-223</td>
</tr>
<tr>
<td>NLSFormatCurrency</td>
<td>2-223</td>
</tr>
<tr>
<td>NLSFormatDate</td>
<td>2-225</td>
</tr>
<tr>
<td>NLSFormatNumber</td>
<td>2-226</td>
</tr>
<tr>
<td>NLSFormatTime</td>
<td>2-228</td>
</tr>
<tr>
<td>MBCS Inquiry Procedures</td>
<td>2-230</td>
</tr>
<tr>
<td>MBCharLen</td>
<td>2-230</td>
</tr>
<tr>
<td>MBCurMax</td>
<td>2-231</td>
</tr>
<tr>
<td>MBLen</td>
<td>2-232</td>
</tr>
<tr>
<td>MBLen_Trim</td>
<td>2-233</td>
</tr>
</tbody>
</table>
Chapter 3 POSIX* Functions

POSIX Library Interface ........................................... 3-1
  IPXFARGC .............................................................. 3-2
  IPXFEXITSTATUS ..................................................... 3-3
  IPXFWSTOPSIG ......................................................... 3-3
  IPXFWTERMSIG ......................................................... 3-4
  PXFACCESS .............................................................. 3-5
  PXFAINTGET ............................................................ 3-6
  PXFAINTSET ............................................................ 3-7
  PXFCALLSUBHANDLE ............................................... 3-8
  PXFCHDIR ............................................................... 3-9
  PXFCHMOD ............................................................. 3-10
  PXFCHOWN ............................................................. 3-11
  PXFCLOSE .............................................................. 3-12
  PXFCLOSEDIR ......................................................... 3-12
  PXFCONST .............................................................. 3-13
  PXFCREAT .............................................................. 3-14
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXFDUP</td>
<td>3-15</td>
</tr>
<tr>
<td>PXFDUP2</td>
<td>3-16</td>
</tr>
<tr>
<td>PXFEINTGET</td>
<td>3-16</td>
</tr>
<tr>
<td>PXFEINTSET</td>
<td>3-17</td>
</tr>
<tr>
<td>PXFESTRGET</td>
<td>3-18</td>
</tr>
<tr>
<td>PXFEXECV</td>
<td>3-19</td>
</tr>
<tr>
<td>PXFEXECVE</td>
<td>3-20</td>
</tr>
<tr>
<td>PXFEXECVP</td>
<td>3-21</td>
</tr>
<tr>
<td>PXFEXIT</td>
<td>3-22</td>
</tr>
<tr>
<td>PXFFASTEXIT</td>
<td>3-23</td>
</tr>
<tr>
<td>PXFFFLUSH</td>
<td>3-23</td>
</tr>
<tr>
<td>PXFFGETC</td>
<td>3-24</td>
</tr>
<tr>
<td>PXFFILENO</td>
<td>3-25</td>
</tr>
<tr>
<td>PXFFORK</td>
<td>3-26</td>
</tr>
<tr>
<td>PXFFPUTC</td>
<td>3-27</td>
</tr>
<tr>
<td>PXFFSEEK</td>
<td>3-28</td>
</tr>
<tr>
<td>PXFFSTAT</td>
<td>3-29</td>
</tr>
<tr>
<td>PXFFTELL</td>
<td>3-30</td>
</tr>
<tr>
<td>PXFGETARG</td>
<td>3-31</td>
</tr>
<tr>
<td>PXFGETC</td>
<td>3-32</td>
</tr>
<tr>
<td>PXFGETCWD</td>
<td>3-32</td>
</tr>
<tr>
<td>PXFGETGRGID</td>
<td>3-33</td>
</tr>
<tr>
<td>PXFGETGRNAM</td>
<td>3-34</td>
</tr>
<tr>
<td>PXFGETPWNAM</td>
<td>3-35</td>
</tr>
<tr>
<td>PXFGETPWUID</td>
<td>3-36</td>
</tr>
<tr>
<td>PXFGETSUBHANDLE</td>
<td>3-37</td>
</tr>
<tr>
<td>PXFINTGET</td>
<td>3-38</td>
</tr>
<tr>
<td>PXFINTSET</td>
<td>3-39</td>
</tr>
<tr>
<td>PXFISBLK</td>
<td>3-40</td>
</tr>
<tr>
<td>PXFISCHR</td>
<td>3-41</td>
</tr>
<tr>
<td>PXFISCONST</td>
<td>3-41</td>
</tr>
<tr>
<td>PXFISDIR</td>
<td>3-42</td>
</tr>
<tr>
<td>PXFISFIFO</td>
<td>3-43</td>
</tr>
<tr>
<td>PXFISREG</td>
<td>3-44</td>
</tr>
<tr>
<td>PXFKILL</td>
<td>3-45</td>
</tr>
<tr>
<td>PXFLINK</td>
<td>3-46</td>
</tr>
<tr>
<td>PXFLOCALTIME</td>
<td>3-47</td>
</tr>
<tr>
<td>PXFLSEEK</td>
<td>3-48</td>
</tr>
<tr>
<td>PXFMKDIR</td>
<td>3-49</td>
</tr>
<tr>
<td>PXFMKFIFO</td>
<td>3-50</td>
</tr>
<tr>
<td>PXFOPEN</td>
<td>3-51</td>
</tr>
<tr>
<td>PXFOPENDIR</td>
<td>3-52</td>
</tr>
<tr>
<td>PXFPIPE</td>
<td>3-53</td>
</tr>
<tr>
<td>PXFPUTC</td>
<td>3-53</td>
</tr>
<tr>
<td>PXFREAD</td>
<td>3-54</td>
</tr>
<tr>
<td>PXFREADDR</td>
<td>3-55</td>
</tr>
<tr>
<td>PXFRENAME</td>
<td>3-56</td>
</tr>
<tr>
<td>PXFREWINDDIR</td>
<td>3-57</td>
</tr>
<tr>
<td>PXFRMDIR</td>
<td>3-57</td>
</tr>
<tr>
<td>PXFSIGADDSET</td>
<td>3-58</td>
</tr>
<tr>
<td>PXFSIGDELSET</td>
<td>3-59</td>
</tr>
<tr>
<td>PXFSIGEMPTYSET</td>
<td>3-60</td>
</tr>
<tr>
<td>PXFSIGFILLSET</td>
<td>3-61</td>
</tr>
<tr>
<td>PXFSIGISMEMBER</td>
<td>3-62</td>
</tr>
<tr>
<td>PXFSTAT</td>
<td>3-63</td>
</tr>
<tr>
<td>PXFSTRGET</td>
<td>3-64</td>
</tr>
<tr>
<td>PXFSTRUCTCOPY</td>
<td>3-65</td>
</tr>
<tr>
<td>PXFSTRUCTCREATE</td>
<td>3-66</td>
</tr>
<tr>
<td>PXFSTRUCTFREE</td>
<td>3-67</td>
</tr>
<tr>
<td>PXFUCOMPARE</td>
<td>3-67</td>
</tr>
<tr>
<td>PXFUMASK</td>
<td>3-68</td>
</tr>
<tr>
<td>PXFUNLINK</td>
<td>3-69</td>
</tr>
<tr>
<td>PXFUTIME</td>
<td>3-70</td>
</tr>
<tr>
<td>PXFWAIT</td>
<td>3-71</td>
</tr>
</tbody>
</table>
Chapter 4  QuickWin Library

QuickWin Subroutines and Functions ........................................  4-2
Graphics Procedures ..........................................................  4-5
Graphic Function Descriptions ................................................  4-14
  ARC ...............................................................................  4-14
  ARC_W .........................................................................  4-15
  CLEARSCREEN ..........................................................  4-16
  DISPLAYCURSOR ...........................................................  4-17
  ELLIPSE ........................................................................  4-18
  ELLIPS_W .....................................................................  4-19
  FLOODFILL .....................................................................  4-21
  FLOODFILL_W .............................................................  4-22
  FLOODFILLRGB ............................................................  4-23
  FLOODFILLRGB_W ......................................................  4-24
  GETARCINFO ................................................................  4-25
  GETBKCOLOR ...............................................................  4-27
  GETCOLOR .....................................................................  4-28
  GETCURRENTPOSITION ...............................................  4-29
  GETCURRENTPOSITION_W .............................................  4-30
  GETFILLMASK ..............................................................  4-31
  GETIMAGE .....................................................................  4-32
  GETIMAGE_W ...............................................................  4-33
  GETLINESIZE ................................................................  4-34
  GETPHYSCOORD ...........................................................  4-35
  GETPIXEL .......................................................................  4-36
  GETPIXEL_W ..................................................................  4-37
GETPIXELS ................................................................. 4-38
GETTEXTCOLOR ...................................................... 4-39
GETTEXTPOSITION .................................................. 4-40
GETTEXTWINDOW ..................................................... 4-40
GETVIEWCOORD ..................................................... 4-41
GETVIEWCOORD_W ............................................... 4-42
GETWINDOWCOORD ................................................. 4-43
GETWRITEMODE ...................................................... 4-44
GRSTATUS .............................................................. 4-45
IMAGESIZE .............................................................. 4-46
IMAGESIZE_W ......................................................... 4-47
LINETO ................................................................. 4-48
LINETO_W .............................................................. 4-49
LINETOAR .............................................................. 4-50
LINETOAREX ......................................................... 4-51
LOADIMAGE ............................................................ 4-52
LOADIMAGE_W ....................................................... 4-53
MOVETO ............................................................... 4-54
MOVETO_W ............................................................. 4-55
OUTTEXT ............................................................... 4-56
PIE ....................................................................... 4-56
PIE_W ................................................................. 4-58
POLYGON ............................................................... 4-60
POLYGON_W ........................................................... 4-61
PUTIMAGE ............................................................ 4-63
PUTIMAGE_W ........................................................ 4-65
RECTANGLE .......................................................... 4-68
RECTANGLE_W ....................................................... 4-69
REMAPALLPALETTERGB ......................................... 4-70
REMAPPALETTERGB ............................................... 4-72
SAVEIMAGE .......................................................... 4-74
SAVEIMAGE_W ....................................................... 4-75
SAVEJPEG.......................................................... 4-76
SAVEJPEG_W...................................................... 4-77
SCROLLTEXTWINDOW........................................... 4-78
SETBKCOLOR ...................................................... 4-79
SETCLIPRGN....................................................... 4-80
SETCOLOR .......................................................... 4-81
SETFILLMASK ..................................................... 4-82
SETLINESTYLE.................................................... 4-83
SETPIXEL ........................................................... 4-84
SETPIXEL_W ....................................................... 4-85
SETPIXELS .......................................................... 4-86
SETTEXTCOLOR .................................................... 4-87
SETTEXTPOSITION .............................................. 4-88
SETTEXTWINDOW ................................................ 4-89
SETVIEWORG ..................................................... 4-90
SETVIEWPORT .................................................... 4-91
SETWINDOW ........................................................ 4-92
SETWRITEMODE .................................................. 4-93
WRAPON ............................................................. 4-95
GETCOLORRGB ...................................................... 4-96
GETBKCOLORRGB .................................................. 4-97
GETPIXELRGB ...................................................... 4-99
GETPIXELRGB_W .................................................. 4-101
SETPIXELSRGB .................................................... 4-102
GETPIXELSRGB .................................................... 4-104
SETCOLORRGB ..................................................... 4-105
SETBKCOLORRGB .................................................. 4-107
SETPIXELRGB ...................................................... 4-108
SETPIXELRGB_W .................................................. 4-110
RGBTOINTEGER .................................................... 4-111
INTERGERTORGB .................................................. 4-112

Font Manipulation Functions ...................................... 4-113
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETFONTINFO</td>
<td>4-113</td>
</tr>
<tr>
<td>GETGTEXTTEXTENT</td>
<td>4-115</td>
</tr>
<tr>
<td>OUTGTEXT</td>
<td>4-116</td>
</tr>
<tr>
<td>INITIALIZEFONTS</td>
<td>4-116</td>
</tr>
<tr>
<td>SETFONT</td>
<td>4-117</td>
</tr>
<tr>
<td>SETGTEXTROTATION</td>
<td>4-120</td>
</tr>
<tr>
<td>GETGTEXTROTATION</td>
<td>4-121</td>
</tr>
<tr>
<td>GETTEXTCOLORRGB</td>
<td>4-122</td>
</tr>
<tr>
<td>SETTEXTCOLORRGB</td>
<td>4-123</td>
</tr>
<tr>
<td>QuickWin Compatible Support</td>
<td>4-125</td>
</tr>
<tr>
<td>ABOUTBOXQQ</td>
<td>4-125</td>
</tr>
<tr>
<td>APPENDMENUQQ</td>
<td>4-126</td>
</tr>
<tr>
<td>CLICKMENUQQ</td>
<td>4-129</td>
</tr>
<tr>
<td>DELETEMENUQQ</td>
<td>4-130</td>
</tr>
<tr>
<td>FOCUSQQ</td>
<td>4-131</td>
</tr>
<tr>
<td>GETACTIVEQQ</td>
<td>4-132</td>
</tr>
<tr>
<td>GETEXITQQ</td>
<td>4-133</td>
</tr>
<tr>
<td>GETHWNDQQ</td>
<td>4-135</td>
</tr>
<tr>
<td>GETUNITQQ</td>
<td>4-136</td>
</tr>
<tr>
<td>GETWINDOWCONFIG</td>
<td>4-137</td>
</tr>
<tr>
<td>GETWSIZEQQ</td>
<td>4-139</td>
</tr>
<tr>
<td>INQFOCUSQQ</td>
<td>4-141</td>
</tr>
<tr>
<td>INSERTMENUQQ</td>
<td>4-143</td>
</tr>
<tr>
<td>MESSAGEBOXQQ</td>
<td>4-146</td>
</tr>
<tr>
<td>MODIFYMENUFLAGSQQ</td>
<td>4-148</td>
</tr>
<tr>
<td>MODIFYMENUROUTINEQQ</td>
<td>4-149</td>
</tr>
<tr>
<td>MODIFYMENUSTRINGQQ</td>
<td>4-151</td>
</tr>
<tr>
<td>REGISTERMOUSEEVENT</td>
<td>4-152</td>
</tr>
<tr>
<td>SETACTIVEQQ</td>
<td>4-154</td>
</tr>
<tr>
<td>SETEXITQQ</td>
<td>4-156</td>
</tr>
<tr>
<td>SETMESSAGEQQ</td>
<td>4-157</td>
</tr>
<tr>
<td>SETWINDOWCONFIG</td>
<td>4-159</td>
</tr>
</tbody>
</table>
SETWINDOWMENUQQ ................................................................. 4-162
SETWSIZEQQ ................................................................. 4-163
UNREGISTERMOUSEEVENT ........................................ 4-164
WAITONMOUSEEVENT ..................................................... 4-166
QuickWin Default Menu Support ........................................ 4-168
WINPRINT ........................................................................ 4-169
WINSAVE ................................................................. 4-169
WINEXIT ................................................................. 4-170
WINCOPY ................................................................. 4-170
WINPASTE ................................................................. 4-171
WINSIZETOFIT .......................................................... 4-171
WINFULLSCREEN .......................................................... 4-172
WINSTATE ................................................................. 4-172
WINCASCADEF .......................................................... 4-173
WINTILE ................................................................. 4-174
WINARRANGE ........................................................... 4-174
WININPUT ................................................................. 4-175
WINCLEARPASTE ........................................................ 4-175
WINSTATUS ............................................................... 4-176
WININDEX ............................................................... 4-176
WINUSING ............................................................... 4-177
WINABOUT ............................................................... 4-177
WINSELECTTEXT ........................................................ 4-178
WINSELECTGRAPHICS ............................................... 4-178
WINSELECTALL ........................................................... 4-179
NUL .............................................................................. 4-179
Unknown Functions ........................................................... 4-180
GETACTIVEPAGE .......................................................... 4-180
GETTEXTCURSOR .......................................................... 4-180
GETGTEXTEXVECTOR .................................................... 4-181
GETHANDLEQQ .......................................................... 4-181
GETVIDEOCONFIG .......................................................... 4-182
GETVISUALPAGE......................................................... 4-183
REGISTERFONTS ..................................................... 4-183
SELECTPALETTE ....................................................... 4-184
SETACTIVEPAGE ...................................................... 4-184
SETFRAMEWINDOW .................................................... 4-185
DSETGTEXTVECTOR ................................................... 4-185
SETSTATUSMESSAGE .................................................. 4-186
SETTEXTCURSOR ....................................................... 4-186
SETTEXTFONT .......................................................... 4-187
SETTEXTROWS .......................................................... 4-187
SETVIDEOMODE ....................................................... 4-188
SETVIDEOMODEROWS ............................................... 4-188
SETVISUALPAGE ....................................................... 4-189
UNREGISTERFONTS .................................................... 4-189

Access to Windows* Handles for QuickWin Components .. 4-190
GETHANDLEFRAMEQQ ................................................ 4-190
GETHANDLECLIENTQQ ............................................... 4-190
GETHANDLECHILDQQ ............................................... 4-191
UNUSEDQQ ............................................................. 4-191

Index
About This Manual

This manual describes the intrinsic, portability, POSIX*, and QuickWin library functions and procedures of the Intel® Fortran libraries for both IA-32 and Itanium® architectures. Wherever the interface is different for the architectures, the difference is described. If not different (default), the description is applicable to two architectures.

The intrinsic, portability, and POSIX libraries documented in this manual can be used for both Windows* and Linux* platforms, the differences, if any, are explained.

For managing and linking libraries with Intel Fortran Compiler, see “Libraries” section in the Intel® Fortran Compiler User's Guide.

This manual is organized as follows:

Chapter 1       Describes the Intel Fortran intrinsic functions
Chapter 2       Describes the portability functions
Chapter 3       Describes the POSIX functions
Chapter 4       Describes QuickWin run-time library functions

Related Publications

The following documents provide additional information relevant to the Intel Fortran 95 Language:

The following documents provide additional information relevant to the Intel Fortran Compiler:


• *Fortran 90/95 for Scientists and Engineers*, S. Chapman; McGraw-Hill

• For Win32-specific information, see the documentation included with the *Microsoft Win32 Software Development Kit*.

• For Microsoft Fortran PowerStation 32 information, see the documentation included with the *Microsoft Fortran Powerstation 32 Development System for Windows NT, Version 1.0*.

Information about the target architecture is available from Intel and from most technical bookstores. Some helpful titles are:

• *Intel® Fortran Programmer’s Reference*

• *Intel® Fortran Compiler User’s Guide*

• *Intel® C/C++ Compiler User’s Guide*


• *Intel Processor Identification with the CPUID Instruction*, doc number 241618

Most Intel documents are also available from the Intel Corporation web site at developer.intel.com

### Notational Conventions

This manual uses the following conventions:

- **This type style** indicates an element of syntax, a reserved word, a keyword, a filename, computer output, or part of a program example. The text appears in lowercase unless uppercase is significant.

- **THIS TYPE STYLE** Fortran source text appears in upper case.
1 is lowercase letter L in examples. 1 is the number 1 in examples. O is the uppercase O in examples. 0 is the number 0 in examples.

This *type style* indicates the exact characters you type as input.

This *type style* indicates a place holder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the place holder.

*[items]* items enclosed in brackets are options.

*(item | item)* Select only one of the items listed between braces. A vertical bar ( | ) separates the items.

*...* Ellipses indicate that you can repeat the preceding item.

This *type style* indicates an Intel Fortran Language extension format.

This *type style* indicates an Intel Fortran Language extension discussion. Throughout the manual, *extensions* to the ANSI standard Fortran language appear in *this font and color* to help you easily identify when your code uses a non-standard language extension.
Intrinsic Procedures

Intrinsic procedures are built-in functions and subroutines that are available by default to every Fortran 95 program and procedure. (If certain conditions are met, an intrinsic procedure can be made unavailable; see Availability of Intrinsic Procedures.)

This chapter describes the intrinsic procedures provided by Intel® Fortran. All intrinsic procedures defined by the Fortran 95 Standard are supported in Intel Fortran. In addition, Intel Fortran supports other nonstandard intrinsic procedures to extend the language’s functionality; see the section Nonstandard Intrinsic Procedures. Intel Fortran intrinsic procedures are provided in the library libintrins.lib.

The Intel® Fortran Compiler User’s Guide has more detailed information about the libraries that are shipped with Intel Fortran and the other libraries used by the ifl compiler driver, including the portlib and posix libraries. The portlib and posix libraries are described in detail in Chapter 2 and Chapter 3 of this manual.

Overview of Intrinsic Procedures

This section explains the situations under which intrinsic procedures are not available, gives an overview of Intel Fortran intrinsic functions and subroutines, describes the use of the INTRINSIC attribute and statement, and discusses nonstandard intrinsic procedures.
Availability of Intrinsic Procedures

An intrinsic procedure is available in every program unit—except when the intrinsic’s name is defined by the user to have a different meaning, such as a representing a user-defined procedure, variable, or constant.

User-defined procedures always take precedence over intrinsic procedures of the same name when the user-defined procedure’s definition is visible. This happens, for example, when the user-defined procedure has an explicit interface, is in an EXTERNAL statement, or is a statement function.

Both a user-defined procedure and an intrinsic may have the same name if the user-defined procedure is used to EXTEND a generic intrinsic and the argument types differ.

An intrinsic function is not available when the function’s name has been given the EXTERNAL attribute.

Intrinsic Subroutines and Functions

Intrinsic procedures include both intrinsic functions and intrinsic subroutines.

An intrinsic subroutine is invoked by the CALL statement and can return values through arguments passed to it. An intrinsic function is referenced as part of an expression and upon evaluation returns a value (the “function value”), which is used in the expression.

Intrinsic procedures are identified as either functions or subroutines as part of their “class” identification.

Intrinsic subroutines can be further classified as either elemental or nonelemental subroutines.

Intrinsic functions include the following classes:

- Elemental Functions
- Inquiry Functions
- Transformational Functions
An intrinsic function can be referenced by either a generic name, a specific name, or both; for details see the section “Generic and Specific Intrinsic Function Names”.

Each intrinsic’s class is noted in the list of intrinsic specifications later in this chapter (see “Intrinsic Procedure Specifications”).

Intrinsic Subroutines

Subroutine references are made by means of the CALL statement. Any values returned by an intrinsic subroutine are provided through the subroutine’s argument(s).

The sample code segment that follows calls the intrinsic subroutine DATE_AND_TIME to get real-time clock and date data, which DATE_AND_TIME returns by means of the argument Dtime.

```
INTEGER Dtime(8)
CALL DATE_AND_TIME(VALUES=Dtime)
PRINT *, Dtime(1) ! print the year
```

Elemental and Nonelemental Subroutines

Intrinsic subroutines include both elemental and nonelemental subroutines. MVBITS is the only elemental subroutine. All other intrinsic subroutines are nonelemental.

MVBITS is elemental in that it allows arrays to be used as arguments in the same way that scalar arguments are specified. It has all scalar dummy arguments but permits conformable arrays to be passed as actual arguments. The effect is as if the scalar form of the subroutine were called for each corresponding element of the actual argument arrays supplied.

Intrinsic Functions

A function differs from a subroutine in that a function returns a value, and a function reference is part of an expression (not a complete statement). Each function returns its result as the function value. This value is used in the expression that references the function.
A function reference may occur wherever an expression is allowed. For instance, intrinsic functions may be used in the right-hand side of assignment statements, as arguments to procedures, in output lists, and elsewhere.

In the following segment of code, the $\text{SIN}$ intrinsic function is evaluated, then its result is printed:

```fortran
Ar = N*Pi/180  ! angle in radians
PRINT *, SIN(Ar)  ! print sine of angle
```

The statement below assigns the product of $Y$ and the sine of $X$ to the variable $Sxy$:

```fortran
Sxy = SIN(X) * Y
```

### Generic and Specific Intrinsic Function Names

There are two varieties of intrinsic function names: generic names and specific names. Each intrinsic function has either a generic name, one or more specific names, or both generic and specific names. If both a generic and specific name exist for an intrinsic function, either may be used to invoke it.

The “Generic and Specific Intrinsic Summary” section later in this chapter lists a summary of generic intrinsic functions and their corresponding specific routines (see Table 1-3).

When you reference a generic intrinsic name, the data type of the actual arguments determine which specific intrinsic is invoked. A reference to a specific intrinsic name requires the intrinsic’s actual arguments to be of a certain data type.

For instance, the generic intrinsic function $\text{ABS}$ can accept arguments of any numeric type. However, a specific version of the intrinsic, $\text{DABS}$, can accept only double precision arguments.

An intrinsic name can be both generic and specific. For example, as shown in Table 1-3, when the intrinsic procedure $\text{SIN}$ is called with a double precision argument the specific function $\text{DSIN}$ is invoked. When $\text{SIN}$ is called with a $\text{REAL}$ argument, however, the specific function $\text{SIN}$ is invoked.
Using a generic name can, in general, simplify the referencing of intrinsic functions, because the generic name can be specified for multiple types of arguments.

**Elemental Functions**

**NOTE.** Some command-line options specify different default data type sizes and can cause different or invalid intrinsic procedure references. For details see the section “Data type sizes and command-line options”.

Elemental intrinsic functions allow arrays to be used as arguments in the same way that scalar arguments are specified.

An elemental function that is called with all scalar dummy arguments delivers a scalar result. Calling an elemental function with conformable array arguments, however, results in a conformable array result. The effect is as if a scalar form of the function were called for each corresponding element of the actual argument arrays supplied.

If both array and scalar arguments are specified to an elemental function, each scalar is treated as an array in which all elements have the scalar value. The “scalar array” is conformable with the array arguments.

The following segment of code illustrates how the elemental intrinsic function ABS can be called with both scalar arguments (such as N) and array arguments (such as X).

```fortran
INTEGER N, Nabs
REAL X(5), Xabs(5)
N = -5
Nabs = ABS(N)
X = (/ -4.5, 5.2, -3.9, -1.1, 8.7 /)
Xabs = ABS(X)
```

After the calls to `ABS`, `Nabs` has the value 5, and the array `Xabs` has the value `[4.5 5.2 3.9 1.1 8.7]`.

**Inquiry Functions**

Inquiry intrinsic functions return information based on their arguments’ properties (and not its arguments’ values).
The following statements illustrate how the `SIZE` inquiry function returns information about a property of the array `A`:

```fortran
REAL A(3:9, 4:10)
INTEGER SizA
SizA = SIZE(A)
```

The `SIZE` intrinsic function returns either the extent of an array along one dimension or the total number of elements in the array. Because `SIZE` is an inquiry intrinsic function, only the array’s properties, and not the values of the elements of the array, are considered.

Following the call to `SIZE`, the variable `SizA` has a value of 49; that is, the total number of elements in array `A` is 49.

**Transformational Functions**

Transformational intrinsic functions include all functions that are not elemental and are not inquiry functions.

In general, transformational functions require at least one array argument, and return either a scalar or array result based on actual arguments that cannot be evaluated elementally. Often, an array result will be of a different shape than the argument(s).

The following code segment makes use of the `ANY` and `ALL` transformational intrinsics.

```fortran
INTEGER(4) A(5), B(5)
LOGICAL L1, L2
A = (/3, 5, 7, 9, 9/)
B = (/3, 4, 7, 8, 9/)
L1 = ANY(A .EQ. B)
L2 = ALL(A .EQ. B)
```

The `ANY` and `ALL` intrinsics functions determine whether any value or all values are `.TRUE.` along dimensions of a logical array.

In the above code, the actual argument passed to `ANY` and `ALL` is a five-element logical array. This array, which is the result of the expression `A .EQ. B`, has the value `[.TRUE., .FALSE., .TRUE., .FALSE., .TRUE.]`. 
After the above statements are evaluated, the logical variable $L_1$ has the value `.TRUE.` (at least one element of $A$ is equal to the corresponding element of $B$) and $L_2$ has the value `.FALSE.` (not all elements of $A$ are equal to the corresponding elements of $B$).

**INTEGRISTIC Attribute and Statement**

Both the **INTEGRISTIC** attribute and the **INTEGRISTIC** statement specify that a name is a specific or generic name of an intrinsic procedure. The **INTEGRISTIC** and **EXTERNAL** attributes are mutually exclusive.

The **INTEGRISTIC** attribute and statement typically are used either to document procedures that are intrinsic or to pass intrinsic procedures as actual arguments.

Functions may be declared intrinsic either in an **INTEGRISTIC** statement or in a type declaration statement using the **INTEGRISTIC** attribute. Such a declaration may occur only once per name.

Intrinsic subroutine names can be declared intrinsic only by using the **INTEGRISTIC** statement. The **INTEGRISTIC** attribute cannot be applied to a subroutine because subroutine names cannot appear in type statements.

**Documenting Intrinsic Procedures**

The **INTEGRISTIC** attribute and statement can be used as documentation techniques to indicate that a name is that of an intrinsic procedure.

This can be useful to other people who will use your code, especially in code where you invoke nonstandard intrinsic procedures.

**Intrinsic Procedures as Actual Arguments**

Intrinsic procedures may be passed as actual arguments to subroutines and functions. Only the names of specific intrinsic procedures may be passed, and the compiler must know that the name being passed is that of an intrinsic.
When an intrinsic procedure is used as an actual argument and its name does not appear elsewhere in the same scoping unit, the intrinsic must be declared intrinsic. This can be done with the INTRINSIC attribute (for intrinsic functions) or the INTRINSIC statement (for subroutines and functions).

For example, if the statement

```fortran
CALL My_Subroutine(QSIN)
```

appears in a program unit and no other occurrence of QSIN appears, the compiler assumes that QSIN is a variable and is not the specific name of the intrinsic function SIN. For this reason, QSIN must be declared intrinsic to ensure that the intrinsic function is passed.

**NOTE.** *Some intrinsic procedures can never be used as actual arguments.*

The following example code shows how the SIN and COS intrinsic functions can be passed to the user-written subroutine *My_Subroutine*.  

```fortran
PROGRAM Example
REAL(4), INTRINSIC :: SIN, COS
CALL My_Subroutine(SIN)
CALL My_Subroutine(COS)
END

SUBROUTINE My_Subroutine(TrigRtn)
REAL(4), EXTERNAL :: TrigRtn
REAL(4), PARAMETER :: Pi=3.1415926
INTEGER I
DO I=0, 360, 45
   ! Convert degrees to radians (I*Pi/180) and call the intrinsic routine passed as TrigRtn.
   WRITE(6, 100) I," degrees ",TrigRtn(I*Pi/180)
END DO
```

**NOTE.** *Some intrinsic procedures can never be used as actual arguments.*
100 FORMAT (I4, A9, F12.8)
END

Using the INTRINSIC attribute, both functions are declared intrinsic in a type declaration statement in the main program unit.

Similarly, in the subroutine, the EXTERNAL attribute specifies that the dummy argument TrigRtn is the name of a function, not the name of a data object.

Nonstandard Intrinsic Procedures

In addition to supporting all intrinsic procedures defined by the Fortran 95 Standard, Intel Fortran provides additional intrinsic procedures that are nonstandard. The nonstandard intrinsics, like the Standard intrinsics, are part of the Intel Fortran language and are not selected or unselected by command-line options.

For a list of nonstandard intrinsic routine names, see Table 1-3. All intrinsic procedures, including all nonstandard procedures, are described in the section “Intrinsic Procedure Specifications”.

The nonstandard intrinsics are included to provide additional functionality not defined in the Standard, to provide compatibility with other Fortran 95 implementations, and to provide specific routines for data types beyond those in the Standard.

NOTE. Using Intel-supplied nonstandard intrinsic procedures in your code may limit its portability. Other vendors’ compilers may not support Intel-supplied nonstandard intrinsics.

In Intel Fortran, nonstandard intrinsic procedures are supported in the same manner as Standard intrinsics; that is, the routines’ generic property, types, and dummy argument attributes are known to the Intel compiler.

The INTRINSIC statement allows a non-Intel Fortran compiler, which may not support Intel nonstandard intrinsics, to immediately indicate if the marked procedures are not recognized as an intrinsic routine.
Data Representation Models

The Fortran 95 Standard specifies data representation models that suggest how data are represented in the computer and how computations are performed on the data. The computations performed by some Fortran 95 intrinsic functions are described in terms of these models.

There are three data representation models in Fortran 95:

- **The Bit Model**
- **The Integer Number System Model**
- **The Real Number System Model**

In a given implementation the model parameters are chosen to match the implementation as closely as possible. However, an exact match is not required and the model does not impose any particular arithmetic on the implementation.

Data Representation Model Intrinsics

Several intrinsic functions provide information about the three data representation models. These intrinsic are listed in Table 1-1.

<table>
<thead>
<tr>
<th>Intrinsic function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_SIZE(I)</td>
<td>Number of bits in an integer of the kind of I (I is an object, not a kind number)</td>
</tr>
<tr>
<td>DIGITS(X)</td>
<td>Base digits of precision in integer or real model for X</td>
</tr>
<tr>
<td>EPSILON(X)</td>
<td>Small value compared to 1 in real model for X</td>
</tr>
<tr>
<td>EXPONENT(X)</td>
<td>Real model exponent value for X</td>
</tr>
<tr>
<td>FRACTION(X)</td>
<td>Real model fraction value for X</td>
</tr>
<tr>
<td>HUGE(X)</td>
<td>Largest model number in integer or real model for X</td>
</tr>
<tr>
<td>MAXEXPONENT(X)</td>
<td>Maximum exponent value in real model for X</td>
</tr>
<tr>
<td>MINEXPONENT(X)</td>
<td>Minimum exponent value in real model for X</td>
</tr>
</tbody>
</table>

continued
The Bit Model

The bit model interprets a nonnegative scalar data object $a$ of type integer as a sequence of binary digits (bits), based upon the model

$$a = \sum_{k=0}^{n-1} b_k 2^k$$

where $n$ is the number of bits, given by the intrinsic function $\text{BIT\_SIZE}$ and each $b_k$ has a bit value of 0 or 1. The bits are numbered from right to left beginning with 0.

The Integer Number System Model

The integer number system is modeled by

$$i = s \sum_{k=0}^{q-1} d_k r^k$$
where

\( i \) is the integer value

\( s \) is the sign (+1 or –1)

\( r \) is the radix given by the intrinsic function \texttt{RADIX} (always 2 for Intel® systems)

\( q \) is the number of digits (integer greater than 0), given by the intrinsic function \texttt{DIGITS}

\( d_k \) is the \( k \)th digit and is an integer \( 0 \leq d_k < r \). The digits are numbered left to right, beginning with 1.

**The Real Number System Model**

The real number system is modeled by

\[
x = s b^e \sum_{k=1}^{p} f_k b^k
\]

where

\( x \) is the real value

\( s \) is the sign (+1 or –1)

\( b \) is the base (real radix) and is an integer greater than 1, given by the intrinsic function \texttt{RADIX} (always 2 for Intel systems)

\( e \) is an integer between some minimum value \((lmin)\) and maximum value \((lmax)\), given by the intrinsic functions \texttt{MINEXPONENT} and \texttt{MAXEXPONENT}

\( p \) is the number of mantissa digits and is an integer greater than 1, given by the intrinsic function \texttt{DIGITS}

\( f_k \) is the \( k \)th digit and is an integer \( 0 \leq f_k < b \), but \( f_1 \) may be zero only if all the \( f_k \) are zero. The digits are numbered left to right, beginning with 1.
Functional Categories of Intrinsic Procedures

A listing of intrinsic procedures, ordered alphabetically by category, appears in Table 1-2. More complete information on the individual intrinsic procedures is provided in the section “Intrinsic Procedure Specifications”.

<table>
<thead>
<tr>
<th>Category</th>
<th>Intrinsic Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array construction</td>
<td>MERGE, PACK, SPREAD, UNPACK</td>
</tr>
<tr>
<td>Array inquiry</td>
<td>ALLOCATED, LBOUND, SHAPE, SIZE, UBOUND</td>
</tr>
<tr>
<td>Array location</td>
<td>MAXLOC, MINLOC</td>
</tr>
<tr>
<td>Array manipulation</td>
<td>CSHIFT, EOSSHIFT, TRANSPOSE</td>
</tr>
<tr>
<td>Array reduction</td>
<td>ALL, ANY, COUNT, MAXVAL, MINVAL, PRODUCT, SUM</td>
</tr>
<tr>
<td>Array reshape</td>
<td>RESHAPE</td>
</tr>
<tr>
<td>Bit inquiry</td>
<td>BIT_SIZE</td>
</tr>
<tr>
<td>Bit manipulation</td>
<td>BTEST, IAND, IBCLR, IBITS, IBSET, IEOR, IOR, ISHFT, ISHFTC, MVBITS, NOT</td>
</tr>
<tr>
<td>Character computation</td>
<td>ACHAR, ADJUSTL, ADJUSTR, CHAR, IACHAR, ICHAR, INDEX, LEN_TRIM, LGE, LGT, LLE, LLT, REPEAT, SCAN, TRIM, VERIFY</td>
</tr>
<tr>
<td>Character inquiry</td>
<td>LEN</td>
</tr>
<tr>
<td>Floating-point manipulation</td>
<td>EXPONENT, FRACTION, NEAREST, RRSPACING, SCALE, SET_EXPONENT, SPACING</td>
</tr>
<tr>
<td>Kind</td>
<td>KIND, SELECT_INT_KIND, SELECTED_REAL_KIND</td>
</tr>
<tr>
<td>Logical</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>Mathematical computation</td>
<td>ACOS, ASIN, ATAN, ATAN2, COS, COSH, EXP, LOG, LOG10, SIN, SINH, SQRT, TAN, TANH</td>
</tr>
<tr>
<td>Matrix multiply</td>
<td>MATMUL</td>
</tr>
<tr>
<td>Nonstandard intrinsic procedures</td>
<td>ACOSD, ACOSH, AND, ASIND, ASINH, ATAN2D, ATAND, ATANH, BADDRESS, COSD, DCMPLX, DFLOAT, DNUM, DREAL, FREE, HFIX, IACHAR, IAADDR, IDIM, IJINT, IMAG, INT1, INT2, INT4, INT8, INUM, ISIGN, ISNAN, IXOR, JNUM, LOC, LSHIFT, LSHIFT, MALLOC, MCLOCK, OR, QNUM, QPROD, RNUM, RSHFT, RSHIFT, SIND, TAN, TANH</td>
</tr>
</tbody>
</table>

Table 1-2 Intrinsic Procedures by Category

continued
As mentioned earlier in the section “Generic and Specific Intrinsic Function Names,” each intrinsic procedure may have a generic name, one or more specific names, or both generic and specific names. All standard and nonstandard generic and specific intrinsic procedures supported by Intel Fortran are summarized in Table 1-3.

Generic and Specific Intrinsic Summary

Table 1-2  Intrinsic Procedures by Category (continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Intrinsic Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric computation</td>
<td>ABS, AIMAG, AINT, ANINT, CEILING, CMPLX, CONJG, DBLE,</td>
</tr>
<tr>
<td></td>
<td>DIM, DPROD, FLOOR, INT, MAX, MIN, MOD, MODULO, NINT,</td>
</tr>
<tr>
<td></td>
<td>REAL, SIGN</td>
</tr>
<tr>
<td>Numeric inquiry</td>
<td>DIGITS, EPSILON, HUGE, MAXEXPONENTS, MINEXPONENTS,</td>
</tr>
<tr>
<td></td>
<td>PRECISION, RADIX, RANGE, TINY</td>
</tr>
<tr>
<td>Pointer inquiry</td>
<td>ASSOCIATED</td>
</tr>
<tr>
<td>Prefetching</td>
<td>MM_PREFETCH</td>
</tr>
<tr>
<td>Presence inquiry</td>
<td>PRESENT</td>
</tr>
<tr>
<td>Pseudorandom number</td>
<td>RANDOM_NUMBER, RANDOM_SEED</td>
</tr>
<tr>
<td>Time</td>
<td>DATE_AND_TIME, SYSTEM_CLOCK</td>
</tr>
<tr>
<td>Transfer</td>
<td>TRANSFER</td>
</tr>
<tr>
<td>Vector multiply</td>
<td>DOT_PRODUCT</td>
</tr>
</tbody>
</table>

Table 1-2  Intrinsic Procedures by Category (continued)
Summary of Generic and Specific Intrinsic Names

Table 1-3 lists a summary of generic and specific intrinsic procedures. The table’s listing is alphabetically ordered.

The class information indicates whether an intrinsic is an extension to the Fortran 95 Standard (‘nonstandard’). These procedures provide nonstandard behavior. Functions whose names and descriptions appear in this color are also extensions to the Fortran 95 standard.

The class information also indicates which intrinsic procedures are subroutines, which are functions, and whether they are specific or generic. When a number appears in parentheses after a type, such as INTEGER, the number represents the KIND value.
Table 1-3  Generic and Specific Intrinsic Procedures

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| ABS                 | Absolute value.  
                     | **Class.** generic elemental function  
                     | **Summary.**  
                     | generic ABS(A) |

```
INTEGER(1) function BABS(A)  
   INTEGER(1) :: A  
 INTEGER(2) function HABS(A)  
   INTEGER(2) :: A  
 INTEGER function IABS(A)  
   INTEGER :: A  
 INTEGER(8) function ABS(A)  
   INTEGER(8) :: A  
 REAL function ABS(A)  
   REAL :: A  
 DOUBLE PRECISION function  
DABS(A)  
   DOUBLE PRECISION :: A  
 REAL function CABS(A)  
   COMPLEX :: A  
 DOUBLE PRECISION function  
CDABS(A)  
   DOUBLE COMPLEX :: A  
 DOUBLE PRECISION function  
ZABS(A)  
   DOUBLE COMPLEX :: A  
 REAL(16) function QABS(X)  
   REAL(16) :: X  
 COMPLEX(16) function CQABS(X)  
   COMPLEX(16) :: X  
end
```
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| ACHAR               | Return character corresponding to ASCII value. **Class.** generic elemental function **Summary.**
|                     | generic ACHAR(I) |
|                     | CHARACTER function ACHAR(I) |
|                     | INTEGER(1) ::I |
|                     | CHARACTER function ACHAR(I) |
|                     | INTEGER(2) ::I |
|                     | CHARACTER function ACHAR(I) |
|                     | INTEGER(4) ::I |
|                     | CHARACTER function ACHAR(I) |
|                     | INTEGER(8) ::I |
|                     | end |
| ACOS                | Arccosine function in radians. **Class.** generic elemental function **Summary.**
|                     | generic ACOS(X) |
|                     | REAL function ACOS(X) |
|                     | REAL ::X |
|                     | DOUBLE PRECISION function DACOS(X) |
|                     | DOUBLE PRECISION::X |
|                     | REAL(16) function QACOS(X) |
|                     | QACOS(X) |
|                     | REAL(16) X |
|                     | end |

**Table 1-3** **Generic and Specific Intrinsic Procedures** (continued)
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOSD</td>
<td>Arcosine function in degrees. Class. generic elemental nonstandard function Summary.</td>
</tr>
<tr>
<td></td>
<td>generic ACOSD(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ACOSD(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(4) ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DACOSD(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QACOSD(X)</td>
</tr>
<tr>
<td></td>
<td>QACOSD(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>ACOSH</td>
<td>Hyperbolic arccosine of radians. Class. generic elemental nonstandard function Summary.</td>
</tr>
<tr>
<td></td>
<td>generic ACOSH(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ACOSH(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(4) ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DACOSH(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QACOSH(X)</td>
</tr>
<tr>
<td></td>
<td>QACOSH(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) ::X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUSTL</td>
<td>Adjust string to left, removing leading blanks.</td>
</tr>
<tr>
<td>Class. generic elemental function</td>
<td></td>
</tr>
<tr>
<td>Summary.</td>
<td></td>
</tr>
<tr>
<td>generic ADJUSTL(STRING)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARACTER function</td>
</tr>
<tr>
<td></td>
<td>ADJUSTL(STRING)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER ::STRING</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>ADJUSTR</td>
<td>Adjust string to right, removing trailing blanks.</td>
</tr>
<tr>
<td>Class. generic elemental function</td>
<td></td>
</tr>
<tr>
<td>Summary.</td>
<td></td>
</tr>
<tr>
<td>generic ADJUSTR(STRING)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARACTER function</td>
</tr>
<tr>
<td></td>
<td>ADJUSTR(STRING)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER ::STRING</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>AIMAG</td>
<td>Imaginary part of a complex number.</td>
</tr>
<tr>
<td>Class. generic elemental function</td>
<td></td>
</tr>
<tr>
<td>Summary.</td>
<td></td>
</tr>
<tr>
<td>generic AIMAG(Z)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL function AIMAG(Z)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX ::Z</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DIMAG(Z)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX ::Z</td>
</tr>
<tr>
<td></td>
<td>QIMAG(Z)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX*32 ::Z</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>AIMAX0</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>AIMIN0</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
</tbody>
</table>

continued
# Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AINT</td>
<td>Truncation to a whole number.</td>
</tr>
<tr>
<td><strong>Class.</strong> generic elemental function</td>
<td></td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td></td>
</tr>
<tr>
<td>generic AINT(A,KIND)</td>
<td></td>
</tr>
<tr>
<td>REAL(4) function AINT(A,KIND)</td>
<td></td>
</tr>
<tr>
<td>REAL(4) ::A</td>
<td></td>
</tr>
<tr>
<td>INTEGER,OPTIONAL ::KIND</td>
<td></td>
</tr>
<tr>
<td>REAL(8) function DINT(A)</td>
<td></td>
</tr>
<tr>
<td>REAL (8)::A</td>
<td></td>
</tr>
<tr>
<td>REAL(16) function QINT(A)</td>
<td></td>
</tr>
<tr>
<td>REAL(16) ::A</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

AJMAX0 see “MAX(A1, A2, A3,...)”

AJMIN0 see “MIN(A1, A2, A3,...)”

AKMAX0 see “MAX(A1, A2, A3,...)”

AKMIN0 see “MIN(A1, A2, A3,...)”

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Determine whether all values are .TRUE. in MASK along dimension DIM.</td>
</tr>
</tbody>
</table>

**Class.** generic transformational function

**Summary.**

generic ALL(MASK,DIM)

! MASK must be array-valued, DIM must be scalar

LOGICAL(1) function

ALL(MASK,DIM)

LOGICAL(1) :: MASK

INTEGER,OPTIONAL::DIM

LOGICAL(2) function

ALL(MASK,DIM)

LOGICAL(2) :: MASK

INTEGER,OPTIONAL::DIM

LOGICAL(4) function

ALL(MASK,DIM)

LOGICAL(4) :: MASK

INTEGER,OPTIONAL::DIM

LOGICAL(8) function

ALL(MASK,DIM)

LOGICAL(8) :: MASK

INTEGER,OPTIONAL::DIM

end

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| ALLOCATED           | Indicate whether an allocated array is currently allocatable. Class: generic inquiry function Summary.  
generic ALLOCATED (ARRAY)  
  ! ARRAY must be an allocatable array  
  LOGICAL function  
  ALLOCATED (ARRAY)  
    INTEGER(1) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY)  
    INTEGER(2) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY)  
    INTEGER(4) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY)  
    INTEGER(8) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY)  
    REAL(4) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY)  
    REAL(8) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY)  
    COMPLEX(4) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY)  
    COMPLEX(8) :: ARRAY  
    LOGICAL function  
    ALLOCATED (ARRAY) |
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| ALLOCATED - continued     | LOGICAL(1) ::ARRAY
|                           | LOGICAL function
|                           | ALLOCATED(ARRAY)                                                          |
|                           | LOGICAL(2) ::ARRAY                                                          |
|                           | LOGICAL function                                                            |
|                           | ALLOCATED(ARRAY)                                                            |
|                           | LOGICAL(4) ::ARRAY                                                          |
|                           | LOGICAL function                                                            |
|                           | ALLOCATED(ARRAY)                                                            |
|                           | LOGICAL(8) ::ARRAY                                                          |
|                           | LOGICAL function                                                            |
|                           | ALLOCATED(ARRAY)                                                            |
|                           | CHARACTER ::ARRAY                                                           |
|                           | LOGICAL function                                                            |
|                           | ALLOCATED(ARRAY)                                                            |
|                           | DERIVED_TYPE ::ARRAY                                                        |

end

<table>
<thead>
<tr>
<th>ALOG</th>
<th>see “LOG(X)”</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOG10</td>
<td>see “LOG10(X)”</td>
</tr>
<tr>
<td>AMAX0</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>AMAX1</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>AMIN0</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>AMIN1</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>AMOD</td>
<td>see “MOD(A, P)”</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **AND**             | Bitwise AND.  
Class. generic elemental nonstandard function  
Summary.  
generic AND(I,J)  

| INTEGER(1) function AND(I,J)  
INTEGER(1) ::I,J  
| INTEGER(2) function AND(I,J)  
INTEGER(2) ::I,J  
| INTEGER(4) function AND(I,J)  
INTEGER(4) ::I,J  
| INTEGER(8) function AND(I,J)  
INTEGER(8) ::I,J  |

end |

| **ANINT**            | Nearest whole number.  
Class. generic elemental function  
Summary.  
generic ANINT(A,KIND)  

| REAL(4) function ANINT(A,KIND)  
REAL(4) ::A  
| INTEGER,OPTIONAL ::KIND  
| REAL(8) function DNINT(A)  
REAL(8) ::A  
| REAL(16) function QNINT(A,KIND)  
REAL(16) :: A  |

end |

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY</td>
<td>Determine whether any value is .TRUE. in MASK along dimension DIM.</td>
</tr>
</tbody>
</table>

**Class.** generic transformational function

**Summary.**

generic ANY(MASK,DIM)

! MASK must be array-valued, DIM must be scalar

   LOGICAL(1) function ANY(MASK,DIM)
   LOGICAL(1) :: MASK;
   INTEGER,OPTIONAL::DIM

   LOGICAL(2) function ANY(MASK,DIM)
   LOGICAL(2) :: MASK;
   INTEGER,OPTIONAL::DIM

   LOGICAL(4) function ANY(MASK,DIM)
   LOGICAL(4) :: MASK;
   INTEGER,OPTIONAL::DIM

   LOGICAL(8) function ANY(MASK,DIM)
   LOGICAL(8) :: MASK;
   INTEGER,OPTIONAL::DIM

end

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIN</td>
<td>Arcsine function in radians.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td></td>
<td>generic ASIN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ASIN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL :: X</td>
</tr>
<tr>
<td></td>
<td>DASIN(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X</td>
</tr>
<tr>
<td></td>
<td>QASIN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>ASIND</td>
<td>Arcsine function in degrees.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td></td>
<td>generic ASIND(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ASIND(X)</td>
</tr>
<tr>
<td></td>
<td>REAL :: X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DASIND(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X</td>
</tr>
<tr>
<td></td>
<td>QASIND(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ASINH** | Hyperbolic arcsine of radians.  
**Class.** generic elemental nonstandard function  
**Summary.**  
 generic ASINH(X)  
 REAL function ASINH(X)  
 REAL :: X  
 DOUBLE PRECISION function DASINH(X)  
 DOUBLE PRECISION :: X  
 REAL(16) function QASINH(X)  
 REAL(16) :: X |
| **ASSOCIATED** | Return association status of pointer or indicate if pointer is associated with a target.  
**Class.** generic inquiry function  
**Summary.**  
 generic ASSOCIATED(POINTER,TARGET)  
 ! POINTER must be a pointer.  
 ! TARGET is optional.  
 ! TARGET must be a pointer or target.  
 ! TARGET may be of any type, including derived type.  
 LOGICAL function ASSOCIATED(POINTER,TARGET) |

---

Continued...
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATAN</strong></td>
<td>Arctangent function in radians.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic ATAN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ATAN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DATAN(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QATAN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td><strong>ATAN2</strong></td>
<td>Arctangent function in radians.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic ATAN2(Y,X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ATAN2(Y,X)</td>
</tr>
<tr>
<td></td>
<td>REAL ::Y,X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DATAN2(Y,X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::Y,X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QATAN2(Y,X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: Y,X</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATAND</td>
<td>Arctangent function in degrees.</td>
</tr>
<tr>
<td>Class.</td>
<td>generic elemental nonstandard function</td>
</tr>
<tr>
<td>Summary.</td>
<td>generic ATAN2D(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ATAND(X)</td>
</tr>
<tr>
<td></td>
<td>REAL ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DATAND(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QATAN2D(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>ATAN2D</td>
<td>Arctangent function in degrees.</td>
</tr>
<tr>
<td>Class.</td>
<td>generic elemental nonstandard function</td>
</tr>
<tr>
<td>Summary.</td>
<td>generic ATAN2D(Y,X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ATAN2D(Y,X)</td>
</tr>
<tr>
<td></td>
<td>REAL ::Y,X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DATAN2D(Y,X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::Y,X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QATAN2D(Y,X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: Y,X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATANH</strong></td>
<td>Hyperbolic arctangent.</td>
</tr>
<tr>
<td>Class.</td>
<td>generic elemental nonstandard function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic ATANH(X)</td>
</tr>
<tr>
<td>REAL function ATANH(X)</td>
<td></td>
</tr>
<tr>
<td>REAL :: X</td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION function DATANH(X)</td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION :: X</td>
<td></td>
</tr>
<tr>
<td>REAL(16) function QATANH(X)</td>
<td></td>
</tr>
<tr>
<td>REAL(16) :: X</td>
<td>end</td>
</tr>
</tbody>
</table>

**BABS** see ABS

**BADDRESS** Return the address of the argument.  Class. generic inquiry nonstandard function  Summary.  INTEGER function BADDRESS(X)  ! X may be of any type, including any derived type.

**BBCLR** see IBCLR

**BBITS** see IBITS

**BBSET** see IBSET

**BBTEST** see BTTEST

**BDIM** see DIM

**BIAND** see IAND

**BIEOR** see IEOR

**BIOR** see IOR

**BITEST** see BTTEST

---

Table 1-3 Generic and Specific Intrinsic Procedures (continued)
### Table 1-3   Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_SIZE</td>
<td>Return the number of bits in an integer.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic inquiry function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td>generic BIT_SIZE(I)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function</td>
</tr>
<tr>
<td></td>
<td>BIT_SIZE(I)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::I</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>BIT_SIZE(I)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::I</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>BIT_SIZE(I)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::I</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>BIT_SIZE(I)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::I</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>BIXOR</td>
<td>see IXOR</td>
</tr>
<tr>
<td>BJTEST</td>
<td>see BTEST</td>
</tr>
<tr>
<td>BKTEST</td>
<td>see BTEST</td>
</tr>
<tr>
<td>BMOD</td>
<td>see &quot;MOD(A, P)&quot;</td>
</tr>
<tr>
<td>BMVBITS</td>
<td>see MVBITS</td>
</tr>
<tr>
<td>BNOT</td>
<td>see NOT</td>
</tr>
<tr>
<td>BSHFT</td>
<td>see ISHFT</td>
</tr>
<tr>
<td>BSHFTC</td>
<td>see BSHFTC</td>
</tr>
<tr>
<td>BSIGN</td>
<td>see SIGN</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| BTEST               | Bit test of an integer value. **Class.** generic elemental function **Summary.**

```
generic BTEST(I,POS)
  LOGICAL(1) function BBTEST(I,POS)
    INTEGER(1) :: I,POS
    BBTEST(I,POS)
  LOGICAL(2) function BITEST(I,POS)
    INTEGER(2) :: I,POS
    BITEST(I,POS)
  LOGICAL(2) function HTEST(I,POS)
    INTEGER(2) :: I,POS
    HTEST(I,POS)
  LOGICAL(4) function BJTEST(I,POS)
    INTEGER(4) :: I,POS
    BJTEST(I,POS)
  LOGICAL(8) function BKTEST(I,POS)
    INTEGER(8) :: I,POS
    BKTEST(I,POS)
end
```

CABS see ABS
CCOS see COS
CDABS see ABS
CDCOS see COS
CDEXP see EXP
CDLOG see “LOG(X)”
CDSIN see SIN
CDSQRT see SQRT

**Table 1-3 Generic and Specific Intrinsic Procedures (continued)**
## Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEILING</td>
<td>Return next larger integer.</td>
</tr>
</tbody>
</table>

**Class.** generic elemental function

**Summary.**

generic CEILING(A,KIND)

    REAL(4) function CEILING(A,KIND)
    REAL(4) ::A
    REAL(4),OPTIONAL ::KIND
    DOUBLE PRECISION function
    CEILING(A, KIND)
    DOUBLE PRECISION ::A
    DOUBLE PRECISION, OPTIONAL ::KIND

    REAL(16) function CEILING(A)
    REAL(16) ::A
    REAL(16), OPTIONAL :: KIND

end

| CEXP                 | see EXP |

---

continued
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>Return integer corresponding to character value.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td></td>
<td>generic CHAR(I,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function</td>
</tr>
<tr>
<td></td>
<td>CHAR(I,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::I;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>CHAR(I,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::I;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>CHAR(I,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::I;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>CHAR(I,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::I;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

CLOG  
see “LOG(X)”  
continued
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPLX</td>
<td>Convert to complex type.</td>
</tr>
</tbody>
</table>

**Class.** generic elemental function  

**Summary.**

generic CMPLX(X,Y,KIND)

    COMPLEX function
    CMPLX(X,Y,KIND)
    INTEGER(1) ::X;
    INTEGER(1),OPTIONAL ::Y
    INTEGER,OPTIONAL ::KIND
    COMPLEX function
    CMPLX(X,Y,KIND)
    INTEGER(2) ::X;
    INTEGER(2),OPTIONAL ::Y
    INTEGER,OPTIONAL ::KIND
    COMPLEX function
    CMPLX(X,Y,KIND)
    INTEGER(4) ::X;
    INTEGER(4),OPTIONAL ::Y
    INTEGER,OPTIONAL ::KIND
    COMPLEX function
    CMPLX(X,Y,KIND)
    INTEGER(8) ::X;
    INTEGER(8),OPTIONAL ::Y
    INTEGER,OPTIONAL ::KIND
    COMPLEX function
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **CMPLX (continued)** | CMPLX(X,Y,KIND)  
REAL :: X;  
REAL,OPTIONAL :: Y  
INTEGER,OPTIONAL :: KIND  
COMPLEX function CMPLX(X,Y,KIND)  
DOUBLE PRECISION :: X;  
DOUBLE PRECISION,OPTIONAL :: Y  
INTEGER,OPTIONAL :: KIND  
COMPLEX function CMPLX(X,KIND)  
COMPLEX :: X;  
INTEGER,OPTIONAL :: KIND  
COMPLEX function CMPLX(X,KIND)  
DOUBLE COMPLEX :: X;  
INTEGER,OPTIONAL :: KIND  
end |

**CONJG**  
Conjugate of a complex number.  
Class. generic elemental function  
Summary.  
generic CONJG(Z)  
COMPLEX function CONJG(Z)  
COMPLEX :: Z  
DOUBLE COMPLEX function DCONJG(Z)  
DOUBLE COMPLEX :: Z  
COMPLEX(16) function QCONJG(Z)  
COMPLEX(16) :: Z  
end  
continued
### Intrinsic Procedures

#### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **COS**             | Cosine function in radians.  
Class. **generic elemental function**  
Summary.  
generic **COS(X)**  
REAL function **COS(X)**  
REAL :: **X**  
DOUBLE PRECISION function **DCOS(X)**  
DOUBLE PRECISION :: **X**  
COMPLEX function **CCOS(X)**  
COMPLEX :: **X**  
DOUBLE COMPLEX function **CDCOS(X)**  
DOUBLE COMPLEX :: **X**  
DOUBLE COMPLEX function **ZCOS(X)**  
DOUBLE COMPLEX :: **X**  
REAL(16) function **QCOS(X)**  
REAL(16) :: **X**  
COMPLEX(16) function **CQCOS(X)**  
COMPLEX(16) :: **X**  
end |
| **COSD**            | Cosine function in degrees.  
Class. **generic elemental nonstandard function**  
Summary.  
generic **COSD(X)**  
REAL function **COSD(X)**  
REAL :: **X**  
DOUBLE PRECISION function **DCOSD(X)**  
DOUBLE PRECISION :: **X**  
end |

---

1-37
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSH</td>
<td>Hyperbolic cosine function.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td></td>
</tr>
<tr>
<td>generic COSH(X)</td>
<td></td>
</tr>
<tr>
<td>REAL function</td>
<td>COSH(X)</td>
</tr>
<tr>
<td>REAL :: X</td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION function</td>
<td>DCOSH(X)</td>
</tr>
<tr>
<td>DOUBLE PRECISION :: X</td>
<td></td>
</tr>
<tr>
<td>REAL(16) function QCOSH(X)</td>
<td></td>
</tr>
<tr>
<td>REAL(16) :: X</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **COUNT** | Count the number of .TRUE. elements of MASK along dimension DIM. 
**Class.** generic transformational function  
**Summary.**  
generic COUNT(MASK,DIM)  
!MASK must be array-valued, DIM must be scalar  
INTEGER function  
COUNT(MASK,DIM)  
LOGICAL(1) :: MASK  
INTEGER,OPTIONAL::DIM  
INTEGER function  
COUNT(MASK,DIM)  
LOGICAL(2) :: MASK  
INTEGER,OPTIONAL::DIM  
INTEGER function  
COUNT(MASK,DIM)  
LOGICAL(4) :: MASK  
INTEGER,OPTIONAL::DIM  
INTEGER function  
COUNT(MASK,DIM)  
LOGICAL(8) :: MASK  
INTEGER,OPTIONAL::DIM  
end

| **CPU_TIME** | Returns the current processor time taking into account the frequency of the processor where the current process is running. To get elapsed CPU_TIME, call the intrinsic twice, once to get the start time, and again to get a finish time, and then subtract start from finish. 
**Class.** generic subroutine  
**Summary.**  
SUBROUTINE CPU_TIME(TIME)  
!TIME must be REAL scalar. Intel Fortran allows REAL(4), REAL(8), REAL(16) or DOUBLE-PRECISION  
continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| CSHIFT              | Circular shift an array expression.  

**Class.** generic transformational function

**Summary.**

```
generic CSHIFT(ARRAY,SHIFT,DIM)
  ! ARRAY must be array-valued,
  ! DIM must be scalar
  INTEGER(1) function
  CSHIFT (ARRAY,SHIFT,DIM)
    INTEGER(1) ::ARRAY;
    INTEGER :: SHIFT;
    INTEGER,OPTIONAL ::DIM
  INTEGER(2) function
  CSHIFT (ARRAY,SHIFT,DIM)
    INTEGER(2) ::ARRAY;
    INTEGER :: SHIFT;
    INTEGER,OPTIONAL ::DIM
  INTEGER(4) function
  CSHIFT (ARRAY,SHIFT,DIM)
    INTEGER(4) ::ARRAY;
    INTEGER :: SHIFT;
    INTEGER,OPTIONAL ::DIM
  INTEGER(8) function
  CSHIFT (ARRAY,SHIFT,DIM)
    INTEGER(8) ::ARRAY;
    INTEGER :: SHIFT;
    INTEGER,OPTIONAL ::DIM
  REAL function
  CSHIFT (ARRAY,SHIFT,DIM)
    REAL(4) ::ARRAY;
    INTEGER :: SHIFT;
    INTEGER,OPTIONAL ::DIM
```
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSHIFT (continued)</td>
<td>DOUBLE PRECISION function CSHIFT(ARRAY,SHIFT,DIM) DOUBLE PRECISION ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM LOGICAL(1) function CSHIFT(ARRAY,SHIFT,DIM) REAL(16) ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM LOGICAL(1) ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM LOGICAL(2) function CSHIFT(ARRAY,SHIFT,DIM) LOGICAL(2) ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM LOGICAL(4) function CSHIFT(ARRAY,SHIFT,DIM) LOGICAL(4) ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM LOGICAL(8) function CSHIFT(ARRAY,SHIFT,DIM) LOGICAL(8) ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM COMPLEX function CSHIFT(ARRAY,SHIFT,DIM) COMPLEX ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSHIFT (continued)</td>
<td>DOUBLE COMPLEX function CSHIFT(ARRAY, SHIFT, DIM)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) function CSHIFT(ARRAY, SHIFT, DIM)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>CHARACTER function CSHIFT(ARRAY, SHIFT, DIM)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>DERIVED_TYPE function CSHIFT(ARRAY, SHIFT, DIM)</td>
</tr>
<tr>
<td></td>
<td>DERIVED_TYPE ::ARRAY; INTEGER :: SHIFT; INTEGER, OPTIONAL ::DIM</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

- **CSIN** see **SIN**
- **CSQRT** see **SQRT**
- **CTAN** see **TAN**
- **DABS** see **ABS**
- **DACOS** see **ACOS**
- **DACOSD** see **ACOSD**
- **DACSIN** see **ACOSH**
- **DASIN** see **ASIN**
- **DASIND** see **ASIND**
- **DASINH** see **ASINH**
- **DATAN** see **ATAN**

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATAN2</td>
<td>see ATAN2</td>
</tr>
<tr>
<td>DATAN2D</td>
<td>see ATAN2D</td>
</tr>
<tr>
<td>DATAND</td>
<td>see ATAND</td>
</tr>
<tr>
<td>DATANH</td>
<td>see ATANH</td>
</tr>
<tr>
<td>DATE_AND_TIME</td>
<td>Return current system date and time.</td>
</tr>
<tr>
<td></td>
<td>Class. generic subroutine</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>subroutine</td>
</tr>
<tr>
<td></td>
<td>DATE_AND_TIME(DATE, TIME, ZONE, VALUES)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER, OPTIONAL :: DATE</td>
</tr>
<tr>
<td></td>
<td>CHARACTER, OPTIONAL :: TIME</td>
</tr>
<tr>
<td></td>
<td>CHARACTER, OPTIONAL :: ZONE</td>
</tr>
<tr>
<td></td>
<td>CHARACTER, OPTIONAL :: VALUES</td>
</tr>
</tbody>
</table>

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLE</td>
<td>Convert to double precision type.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td></td>
<td>generic DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>REAL ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX ::A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DBLE(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX ::A</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLE (continued)</td>
<td>DOUBLE PRECISION function DBLE(A) COMPLEX(16) ::A end</td>
</tr>
<tr>
<td>DBLEQ</td>
<td>see DBLE</td>
</tr>
</tbody>
</table>

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DCMPLX</strong></td>
<td>Convert to double precision complex type.</td>
</tr>
</tbody>
</table>

**Class:** generic elemental nonstandard function

**Summary:**

generic DCMPLX(X,Y)

```fortran
COMPLEX(8) function DCMPLX(X,Y)
INTEGER(1) :: X;
INTEGER(1), OPTIONAL :: Y
COMPLEX(8) function DCMPLX(X,Y)
INTEGER(2) :: X;
INTEGER(2), OPTIONAL :: Y
COMPLEX(8) function DCMPLX(X,Y)
INTEGER(4) :: X;
INTEGER(4), OPTIONAL :: Y
COMPLEX(8) function DCMPLX(X,Y)
INTEGER(8) :: X;
INTEGER(8), OPTIONAL :: Y
COMPLEX(8) function DCMPLX(X,Y)
REAL(4) :: X;
REAL(4), OPTIONAL :: Y
COMPLEX(8) function DCMPLX(X,Y)
DOUBLE PRECISION :: X;
DOUBLE PRECISION, OPTIONAL :: Y
DOUBLE COMPLEX function DCMPLX(X,Y)
REAL(16) :: X;
REAL(16), OPTIONAL :: Y
```

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCMPLX (continued)</td>
<td>COMPLEX(8) function DCMPLX(X)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(4) :: X</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(8) function DCMPLX(X)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(8) :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>DCONJG</td>
<td>see CONJG</td>
</tr>
<tr>
<td>DCOS</td>
<td>see COS</td>
</tr>
<tr>
<td>DCOSD</td>
<td>see COSD</td>
</tr>
<tr>
<td>DCOSH</td>
<td>see COSH</td>
</tr>
<tr>
<td>DDIM</td>
<td>see DIM</td>
</tr>
<tr>
<td>DDINT</td>
<td>see AINT</td>
</tr>
<tr>
<td>DEXP</td>
<td>see EXP</td>
</tr>
<tr>
<td>DFLOAT</td>
<td>Convert to double precision type.</td>
</tr>
<tr>
<td></td>
<td>Class. generic elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic DFLOAT(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DFLOAT(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DFLOATI(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DFLOATJ(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DFLOATK(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: A</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>DFLOATI</td>
<td>see DFLOAT</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFLOTJ</td>
<td>see DFLOAT</td>
</tr>
<tr>
<td>DFLOTK</td>
<td>see DFLOAT</td>
</tr>
<tr>
<td>DIGITS</td>
<td>Return number of significant digits in the model.</td>
</tr>
</tbody>
</table>

**Class.** generic inquiry function

**Summary.**

generic DIGITS(X)

    INTEGER function DIGITS(X)
    INTEGER(1) :: X
    INTEGER function DIGITS(X)
    INTEGER(2) :: X
    INTEGER function DIGITS(X)
    INTEGER(4) :: X
    INTEGER function DIGITS(X)
    INTEGER(8) :: X
    INTEGER function DIGITS(X)
    REAL :: X
    INTEGER function DIGITS(X)
    DOUBLE PRECISION :: X

end
Table 1-3  **Generic and Specific Intrinsic Procedures**  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| DIM                 | Positive difference.  
  **Class.** generic elemental function  
  **Summary.**  
  generic DIM(X,Y)  
  
  INTEGER(1) function BDIM(X,Y)  
  INTEGER(1) ::X,Y  
  INTEGER(2) function HDIM(X,Y)  
  INTEGER(2) ::X,Y  
  INTEGER(4) function IDIM(X,Y)  
  INTEGER(4) ::X,Y  
  INTEGER(8) function KDIM(X,Y)  
  INTEGER(8) ::X,Y  
  REAL function DIM(X,Y)  
  REAL ::X,Y  
  DOUBLE PRECISION function  
  DIM(X,Y)  
  DOUBLE PRECISION ::X,Y  
  end |
| DIMAG               | see IMAG |
| DINT                | see AINT |
| DLOG                | see "LOG(X)" |
| DLOG10              | see "LOG10(X)" |
| DMAX1               | see "MAX(A1, A2, A3,...)" |
| DMIN1               | see "MIN(A1, A2, A3,...)" |
| DMOD                | see "MOD(A, P)" |
| DNINT               | see ANINT |

**continued**
### DNUM

**Class.** specific elemental nonstandard function  
**Summary.**  
DOUBLE PRECISION function DNUM(I)  

| CHARACTER :: I |

### DOT_PRODUCT

**Class.** generic transformational function  
**Summary.**  

**Notes.**  
- **VECTOR_A** must be of numeric type (integer, real, complex) or of logical type. It must be array-valued and of rank one.  
- **VECTOR_B** must be of numeric type if **VECTOR_A** is of numeric type or of logical type if **VECTOR_A** is of type logical. It must have the same shape as **VECTOR_A**.  
If the arguments are of numeric type:  
- If **VECTOR_A** is of type integer or real, the result has value \( \text{SUM}(\text{VECTOR}_A \times \text{VECTOR}_B) \).  
- If **VECTOR_A** is of type complex, the result has value \( \text{CONJG}(\text{VECTOR}_A) \times \text{VECTOR}_B \).  
- If the arguments are of logical type the result has value \( \text{ANY}(\text{VECTOR}_A . \text{AND.} \ \text{VECTOR}_B) \).  

### DPROD

**Class.** generic elemental function  
**Summary.**  

**Notes.**  
- If **VECTOR_A** is of type integer or real, the result has value \( \text{SUM}(\text{VECTOR}_A \times \text{VECTOR}_B) \).  
- If **VECTOR_A** is of type complex, the result has value \( \text{CONJG}(\text{VECTOR}_A) \times \text{VECTOR}_B \).  
- If the arguments are of logical type the result has value \( \text{ANY}(\text{VECTOR}_A . \text{AND.} \ \text{VECTOR}_B) \).  

---

**Table 1-3  Generic and Specific Intrinsic Procedures** (continued)  

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| DNUM                | Convert to double precision.  
**Class.** specific elemental nonstandard function  
**Summary.**  
DOUBLE PRECISION function DNUM(I)  

| CHARACTER :: I |

| DOT_PRODUCT      | Dot product multiplication of numeric or logical vectors.  
**Class.** generic transformational function  
**Summary.**  

generic DOT_PRODUCT(VECTOR_A,VECTOR_B)  

| NOTES. | | VECTOR_A must be of numeric type (integer, real, complex) or of logical type. It must be array-valued and of rank one.  
VECTOR_B must be of numeric type if **VECTOR_A** is of numeric type or of logical type if **VECTOR_A** is of type logical. It must have the same shape as **VECTOR_A**.  
If the arguments are of numeric type:  
- If **VECTOR_A** is of type integer or real, the result has value \( \text{SUM}(\text{VECTOR}_A \times \text{VECTOR}_B) \).  
- If **VECTOR_A** is of type complex, the result has value \( \text{CONJG}(\text{VECTOR}_A) \times \text{VECTOR}_B \).  
- If the arguments are of logical type the result has value \( \text{ANY}(\text{VECTOR}_A . \text{AND.} \ \text{VECTOR}_B) \).  

| DPROD              | Double precision real product.  
**Class.** generic elemental function  
**Summary.**  

generic DPROD(X,Y)  

DOUBLE PRECISION function  
DPROD(X,Y)  

REAL :: X,Y  
end |
Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DREAL</td>
<td>Convert to double precision. Class. generic elemental nonstandard function Summary.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DREAL(A)</td>
<td>generic DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>REAL :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>REAL(16):: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX :: A</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DREAL(A)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) :: A</td>
</tr>
<tr>
<td>end</td>
<td>continued</td>
</tr>
</tbody>
</table>
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIGN</td>
<td>see SIGN</td>
</tr>
<tr>
<td>DSIN</td>
<td>see SIN</td>
</tr>
<tr>
<td>DSIND</td>
<td>see SIND</td>
</tr>
<tr>
<td>DSINH</td>
<td>see SINH</td>
</tr>
<tr>
<td>DSQRT</td>
<td>see SQRT</td>
</tr>
<tr>
<td>DTAN</td>
<td>see TAN</td>
</tr>
<tr>
<td>DTAND</td>
<td>see TAND</td>
</tr>
<tr>
<td>DTANH</td>
<td>see TANH</td>
</tr>
</tbody>
</table>

**EOSHIFT**  
End off shift on an array expression.  
**Class.** generic transformational function  
**Summary.**  
generic EOSHIFT(ARRAY,SHIFT,BOUNDARY,DIM)  
! ARRAY must be array-valued  
! SHIFT must be of type  
integer--see Notes below  
! DIM must be a scalar integer  
! BOUNDARY and DIM are optional  
INTEGER(1) function  
EOSHIFT(ARRAY,SHIFT,BOUNDARY,DIM)  
INTEGER(1) ::ARRAY,  
BOUNDARY  
INTEGER(2) function  
EOSHIFT(ARRAY,SHIFT,BOUNDARY,DIM)  
INTEGER(2) ::ARRAY,  
BOUNDARY  
INTEGER(4) function  
EOSHIFT(ARRAY,SHIFT,BOUNDARY,DIM)  
INTEGER(4) ::ARRAY,  
BOUNDARY  
INTEGER(8) function  
EOSHIFT(ARRAY,SHIFT,BOUNDARY,DIM)  
INTEGER(8) ::ARRAY,  
BOUNDARY

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOSHIFT (continued)</td>
<td>REAL function</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>REAL :: ARRAY, BOUNDARY</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>DOUBLE PRECISION :: ARRAY, BOUNDARY</td>
</tr>
<tr>
<td>LOGICAL(1) function</td>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
</tr>
<tr>
<td>LOGICAL(1) :: ARRAY, BOUNDARY</td>
<td></td>
</tr>
<tr>
<td>LOGICAL(2) function</td>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
</tr>
<tr>
<td>LOGICAL(2) :: ARRAY, BOUNDARY</td>
<td></td>
</tr>
<tr>
<td>LOGICAL(4) function</td>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
</tr>
<tr>
<td>LOGICAL(4) :: ARRAY, BOUNDARY</td>
<td></td>
</tr>
<tr>
<td>LOGICAL(8) function</td>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
</tr>
<tr>
<td>LOGICAL(8) :: ARRAY, BOUNDARY</td>
<td></td>
</tr>
</tbody>
</table>

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOSHIFT (continued)</td>
<td>LOGICAL(8) :: ARRAY, BOUNDARY</td>
</tr>
<tr>
<td></td>
<td>COMPLEX function</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>COMPLEX :: ARRAY, BOUNDARY</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX function</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>DOUBLE COMPLEX :: ARRAY, BOUNDARY</td>
</tr>
<tr>
<td></td>
<td>CHARACTER function</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>CHARACTER :: ARRAY, BOUNDARY</td>
</tr>
<tr>
<td></td>
<td>DERIVED TYPE function</td>
</tr>
<tr>
<td>EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)</td>
<td>DERIVED_TYPE :: ARRAY, BOUNDARY</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

Notes.

SHIFT must be of type integer and must be scalar if ARRAY has rank one; otherwise see the section "EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)" on page 194.

EPSILON

Return positive number that is almost negligible compared to unity in the real number model.

Class. generic inquiry function

Summary.

generic EPSILON(X)

REAL function EPSILON(X)

REAL :: X

DOUBLE PRECISION function EPSILON(X)

DOUBLE PRECISION :: X

end
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **EXP**             | Exponential.  
**Class.** generic elemental function  
**Summary.**  
generic EXP(X)  

```fortran
    REAL function EXP(X)  
        REAL ::X  
    DOUBLE PRECISION function DEXP(X)  
        DOUBLE PRECISION ::X  
    COMPLEX function CEXP(X)  
        COMPLEX ::X  
    DOUBLE COMPLEX function CDEXP(X)  
        DOUBLE COMPLEX ::X  
    DOUBLE COMPLEX function ZEXP(X)  
        DOUBLE COMPLEX ::X  
end
```

| **EXPONENT**        | Return the exponent part of the argument when represented as a model number.  
**Class.** generic elemental function  
**Summary.**  
generic EXPONENT(X)  

```fortran
    INTEGER function EXPONENT(X)  
        REAL ::X  
    INTEGER function EXPONENT(X)  
        INTEGER function EXPONENT(X)  
        DOUBLE PRECISION ::X  
end
```

| **FLOAT**           | see REAL  
| **FLOATI**          | see REAL  
| **FLOATJ**          | see REAL  

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOATK</td>
<td>see REAL</td>
</tr>
</tbody>
</table>
| FLOOR               | Return the greatest integer less than or equal to the argument. **Class.** generic elemental function  
  **Summary.**  
  generic FLOOR(A, KIND)  
  REAL function FLOOR(A, KIND)  
  REAL ::A  
  REAL, OPTIONAL ::KIND  
  DOUBLE PRECISION function  
  FLOOR(A, KIND)  
  DOUBLE PRECISION ::A  
  DOUBLE PRECISION, OPTIONAL ::KIND  
  REAL(16) function FLOOR(A)  
  REAL(16)::A  
  REAL(16), OPTIONAL ::KIND  
  end |
| FRACTION            | Return the fractional part of the model representation of the argument value. **Class.** generic elemental function  
  **Summary.**  
  generic FRACTION(X)  
  REAL function FRACTION(X)  
  REAL ::X  
  DOUBLE PRECISION function  
  FRACTION(X)  
  DOUBLE PRECISION ::X  
  REAL(16) function FRACTION(X)  
  REAL(16) X  
  end |
| FREE(A)             | Frees memory that is currently allocated. **Class.** Elemental nonstandard subroutine. |
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HABS</td>
<td>see ABS</td>
</tr>
<tr>
<td>HBCLR</td>
<td>see IBCLR</td>
</tr>
<tr>
<td>HBITS</td>
<td>see IBITS</td>
</tr>
<tr>
<td>HBSET</td>
<td>see IBSET</td>
</tr>
<tr>
<td>HDIM</td>
<td>see DIM</td>
</tr>
</tbody>
</table>
| HFIX                | Convert to INTEGER(2) type.  
  Class: generic elemental nonstandard function  
  Summary: generic HFIX(A)  
  
  INTEGER(2) function HFIX(A)  
  INTEGER(1) :: A  
  INTEGER(2) function HFIX(A)  
  INTEGER(2) :: A  
  INTEGER(2) function HFIX(A)  
  INTEGER(4) :: A  
  INTEGER(2) function HFIX(A)  
  INTEGER(8) :: A  
  INTEGER(2) function IIDINT(A)  
  DOUBLE PRECISION :: A  
  INTEGER(2) function HFIX(A)  
  COMPLEX :: A  
  INTEGER(2) function HFIX(A)  
  DOUBLE COMPLEX :: A  
  end |
| HIAND               | see IAND    |
| HIEOR               | see IEOR    |
| HIOR                | see IOR     |
| HIXOR               | see IXOR    |
| HMOD                | see “MOD(A, P)” |
| HMVBITS             | see MVBITS  |

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNOT</td>
<td>see NOT</td>
</tr>
<tr>
<td>HSHFT</td>
<td>see ISHFT</td>
</tr>
<tr>
<td>HSHFTC</td>
<td>see ISHFTC</td>
</tr>
<tr>
<td>HSIGN</td>
<td>see SIGN</td>
</tr>
<tr>
<td>HTEST</td>
<td>see BTEST</td>
</tr>
<tr>
<td>HUGE</td>
<td>Return the largest number in the model representing numbers.</td>
</tr>
</tbody>
</table>

**Class.** generic inquiry function

**Summary.**

generic HUGE(X)

    INTEGER(1) function HUGE(X)
    INTEGER(1) :: X
    INTEGER(2) function HUGE(X)
    INTEGER(2) :: X
    INTEGER(4) function HUGE(X)
    INTEGER(4) :: X
    INTEGER(8) function HUGE(X)
    INTEGER(8) :: X
    REAL function HUGE(X)
    REAL :: X
    DOUBLE PRECISION function HUGE(X)
    DOUBLE PRECISION :: X
    REAL(16) function HUGE(X)
    REAL(16) HUGE

    end  

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| IABS                | Integer absolute value.  
**Class.** specific elemental function  
**Summary.**  
generic IABS(A)  

```fortran
INTEGER(1) function IABS(A)  
INTEGER(1) ::A  
INTEGER(2) function IIABS(A)  
INTEGER(2) ::A  
INTEGER(4) function JIABS(A)  
INTEGER(4) ::A  
INTEGER(8) function KIABS(A)  
INTEGER(8) ::A  
end
```
| IACHAR              | Return the position of a character in the ASCII sequence  
**Class.** generic elemental function  
**Summary.**  
generic IACHAR(C)  

```fortran
INTEGER function IACHAR(C)  
CHARACTER ::C  
end
```

---

*Table 1-3  Generic and Specific Intrinsic Procedures* (continued)
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IADDR</td>
<td>Return the address of the argument.</td>
</tr>
<tr>
<td></td>
<td><strong>Class:</strong> specific inquiry nonstandard function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary:</strong> INTEGER function IADDR(X)</td>
</tr>
<tr>
<td></td>
<td>! X may be any one of the following:</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: X</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: X</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: X</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: X</td>
</tr>
<tr>
<td></td>
<td>REAL :: X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X</td>
</tr>
<tr>
<td></td>
<td>COMPLEX :: X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX :: X</td>
</tr>
<tr>
<td></td>
<td>LOGICAL(1) :: X</td>
</tr>
<tr>
<td></td>
<td>LOGICAL(2) :: X</td>
</tr>
<tr>
<td></td>
<td>LOGICAL(4) :: X</td>
</tr>
<tr>
<td></td>
<td>LOGICAL(8) :: X</td>
</tr>
<tr>
<td></td>
<td>CHARACTER :: X</td>
</tr>
<tr>
<td></td>
<td><strong>DERIVED_TYPE</strong> :: X</td>
</tr>
</tbody>
</table>

continued
Intrinsic Procedures

Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAND</td>
<td>Bitwise logical AND.</td>
</tr>
</tbody>
</table>

**Class.** generic elemental function

**Summary.**

```
generic IAND(I,J)

 INTEGER(1) function BIAND(I,J)
 INTEGER(1) ::I,J
 INTEGER(2) function HIAND(I,J)
 INTEGER(2) ::I,J
 INTEGER(2) function IIAND(I,J)
 INTEGER(2) ::I,J
 INTEGER(4) function JIAND(I,J)
 INTEGER(4) ::I,J
 INTEGER(8) function KIAND(I,J)
 INTEGER(8) ::I,J

end
```

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBCLR</td>
<td>Clear a bit to zero.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td></td>
<td>generic IBCLR(I,POS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function</td>
</tr>
<tr>
<td></td>
<td>BBCLR(I,POS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: I, POS</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>HBCLR(I,POS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: I, POS</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>IIBCLR(I,POS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: I, POS</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>JIBCLR(I,POS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: I, POS</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>KIBCLR(I,POS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: I, POS</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

---

*continued*
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBITS</td>
<td>Extract a sequence of bits.</td>
</tr>
</tbody>
</table>

**Class.** generic elemental function  

**Summary.**

```plaintext
generic IBITS(I,POS,LEN)  
   INTEGER(1) function BBITS(I,POS,LEN)  
      INTEGER(1) :: I,POS,LEN  
   INTEGER(2) function HBITS(I,POS,LEN)  
      INTEGER(2) :: I,POS,LEN  
   INTEGER(2) function IIBITS(I,POS,LEN)  
      INTEGER(2) :: I,POS,LEN  
   INTEGER(4) function JIBITS(I,POS,LEN)  
      INTEGER(4) :: I,POS,LEN  
   INTEGER(8) function KIBITS(I,POS,LEN)  
      INTEGER(8) :: I,POS,LEN  
end
```

continued
IBSET

Set a bit to one.

**Class.** generic elemental function

**Summary.**

generic IBSET(I,POS)

    INTEGER(1) function BBSET(I,POS)
    INTEGER(1) :: I,POS
    INTEGER(2) function HBSET(I,POS)
    INTEGER(2) :: I,POS
    INTEGER(2) function IIBSET(I,POS)
    INTEGER(2) :: I,POS
    INTEGER(4) function JIBSET(I,POS)
    INTEGER(4) :: I,POS
    INTEGER(8) function KIBSET(I,POS)
    INTEGER(8) :: I,POS
end

ICHAR

Return ASCII value corresponding to character.

**Class.** generic elemental function

**Summary.**

generic ICHAR(C)

    INTEGER function ICHAR(C)
    CHARACTER :: C
end

---

**Table 1-3  Generic and Specific Intrinsic Procedures (continued)**

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBSET</td>
<td>Set a bit to one.</td>
</tr>
</tbody>
</table>
| **Class.** generic elemental function **Summary.**
| generic IBSET(I,POS) |
| BBSET(I,POS)        | INTEGER(1) function |
| HBSET(I,POS)        | INTEGER(2) function |
| IIBSET(I,POS)       | INTEGER(2) function |
| JIBSET(I,POS)       | INTEGER(4) function |
| KIBSET(I,POS)       | INTEGER(8) function |
| ICHAR               | Return ASCII value corresponding to character. |
| **Class.** generic elemental function **Summary.**
| generic ICHAR(C)    | INTEGER function ICHAR(C) |
| end                 | CHARACTER :: C |

---

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **IDIM**            | Integer positive difference.  
**Class.** specific elemental nonstandard function  
**Summary.**  
generic function DIM(X,Y)  
    INTEGER(1) function BDIM(X,Y)  
    INTEGER(1) ::X,Y  
    INTEGER(2) function IIDIM(X,Y)  
    INTEGER(2) ::X,Y  
    INTEGER(4) function JIDIM(X,Y)  
    INTEGER(4) ::X,Y  
    INTEGER(8) function KIDIM(X,Y)  
    INTEGER(8) ::X,Y  
end |

| **IDINT**           | see INT8 |
| **IDNINT**          | see NINT |
| **IEOR**            | Bitwise exclusive OR.  
**Class.** generic elemental function  
**Summary.**  
generic IEOR(I,J)  
    INTEGER(1) function BIEOR(I,J)  
    INTEGER(1) ::I,J  
    INTEGER(2) function HIEOR(I,J)  
    INTEGER(2) ::I,J  
    INTEGER(2) function IIEOR(I,J)  
    INTEGER(2) ::I,J  
    INTEGER(4) function JIEOR(I,J)  
    INTEGER(4) ::I,J  
    INTEGER(8) function KIEOR(I,J)  
    INTEGER(8) ::I,J  
end |

| **IFIX**            | see INT |
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIABS</td>
<td>see IABS</td>
</tr>
<tr>
<td>IIAND</td>
<td>see IAND</td>
</tr>
<tr>
<td>IIANDR</td>
<td>see IIBCLR</td>
</tr>
<tr>
<td>IIBITS</td>
<td>see IIBITS</td>
</tr>
<tr>
<td>IISET</td>
<td>see IISET</td>
</tr>
<tr>
<td>IIDIM</td>
<td>see IIDIM</td>
</tr>
<tr>
<td>IIDINT</td>
<td>see IIDINT</td>
</tr>
<tr>
<td>IIDNNT</td>
<td>see IIDNNT</td>
</tr>
<tr>
<td>IIEOR</td>
<td>see IIEOR</td>
</tr>
<tr>
<td>IIFIX</td>
<td>see IIFIX</td>
</tr>
<tr>
<td>IINT</td>
<td>see IINT</td>
</tr>
<tr>
<td>IIOR</td>
<td>see IIOR</td>
</tr>
<tr>
<td>IIQINT</td>
<td>see IIQINT</td>
</tr>
<tr>
<td>IIQNNT</td>
<td>see IIQNNT</td>
</tr>
<tr>
<td>IIQINT</td>
<td>see IIQINT</td>
</tr>
<tr>
<td>IISHFT</td>
<td>see IISHFT</td>
</tr>
<tr>
<td>IISHFTC</td>
<td>see IISHFTC</td>
</tr>
<tr>
<td>IISIGN</td>
<td>see IISIGN</td>
</tr>
<tr>
<td>IIXOR</td>
<td>see IIXOR</td>
</tr>
<tr>
<td>IJINT</td>
<td>Convert to INTEGER(2).</td>
</tr>
<tr>
<td></td>
<td>Class: specific elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td>Summary:</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function IJINT(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: A</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **IMAG**            | Imaginary part of a complex number.  
Class. generic elemental nonstandard function  
Summary.  
generic IMAG(Z)  
    REAL function IMAG(Z)  
    COMPLEX ::Z  
    DOUBLE PRECISION function  
    DIMAG(Z)  
    DOUBLE COMPLEX ::Z  
end |
| **IMAX0**           | see “MAX(A1, A2, A3, ...)” |
| **IMAX1**           | see “MAX(A1, A2, A3, ...)” |
| **IMIN0**           | see “MIN(A1, A2, A3, ...)” |
| **IMIN1**           | see “MIN(A1, A2, A3, ...)” |
| **IMOD**            | see “MOD(A, P)” |
| **INDEX**           | Return the starting position of a substring within a string.  
Class. generic elemental function  
Summary.  
generic INDEX(STRING, SUBSTRING, BACK)  
    INTEGER function  
    INDEX(STRING, SUBSTRING, BACK)  
    CHARACTER::STRING  
    CHARACTER::SUBSTRING  
    LOGICAL, OPTIONAL::BACK  
end |
| **ININT**           | see NINT |
| **INOT**            | see NOT |

continued
## Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| INT                 | Convert to integer type.  
  **Class.** generic elemental function  
  **Summary.**  
  generic INT(A,KIND)  
  
  INTEGER(2) function IINT(A)  
  REAL :: A  
  INTEGER function INT(A, KIND)  
  INTEGER(1) :: A;  
  INTEGER,OPTIONAL :: KIND  
  INTEGER function INT(A, KIND)  
  INTEGER(2) :: A  
  INTEGER,OPTIONAL :: KIND  
  INTEGER(4) function IFIX(A)  
  REAL :: A  
  INTEGER(8) function KIFIX(A)  
  REAL :: A  
  INTEGER(8) function KINT(A)  
  REAL :: A  
  INTEGER(4) function JIFIX(A)  
  REAL :: A  
  INTEGER(4) function JINT(A)  
  REAL :: A  
  INTEGER(4) function IDINT(A)  
  DOUBLE PRECISION :: A  
  INTEGER(4) function JIDINT(A)  
  DOUBLE PRECISION :: A  
  INTEGER(2) function IIQINT(A)  
  REAL(16) :: A  
  INTEGER function INT(A, KIND)  
  COMPLEX :: A |
Table 1-3  **Generic and Specific Intrinsic Procedures** (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT (continued)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function INT(A, KIND)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function INT(A,KIND)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: KIND</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INT1</strong></td>
<td>Convert to INTEGER(1) type.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental nonstandard function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td><strong>generic</strong> INT1(A)</td>
</tr>
</tbody>
</table>
|                     | \[
|                   | INTEGER(1) function INT1(A) |
|                   | INTEGER(1) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | INTEGER(2) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | INTEGER(4) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | INTEGER(8) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | REAL :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | DOUBLE PRECISION :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | REAL(16) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | COMPLEX :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | DOUBLE COMPLEX :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | COMPLEX(16) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | LOGICAL(1) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | LOGICAL(2) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | LOGICAL(4) :: A |
|                   | INTEGER(1) function INT1(A) |
|                   | LOGICAL(8) :: A |
|                     | end |

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT2</td>
<td>Convert to INTEGER(2) type.</td>
</tr>
</tbody>
</table>

**Class.** generic elemental nonstandard function

**Summary.**

generic INT2(A)

```fortran
  INTEGER(2) function INT2(A)
  INTEGER(1) :: A
  INTEGER(2) function INT2(A)
  INTEGER(2) :: A
  INTEGER(2) function INT2(A)
  INTEGER(4) :: A
  INTEGER(2) function INT2(A)
  INTEGER(8) :: A
  INTEGER(2) function INT2(A)
  REAL :: A
  INTEGER(2) function INT2(A)
  DOUBLE PRECISION :: A
  INTEGER(2) function INT2(A)
  COMPLEX :: A
  INTEGER(2) function INT2(A)
  DOUBLE COMPLEX :: A
  INTEGER(2) function INT2(A)
  LOGICAL(1) :: A
  INTEGER(2) function INT2(A)
  LOGICAL(2) :: A
  INTEGER(2) function INT2(A)
  LOGICAL(4) :: A
  INTEGER(2) function INT2(A)
  LOGICAL(8) :: A
end
```

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INT4</strong></td>
<td>Convert to INTEGER(4) type.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td></td>
<td>generic INT4(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>REAL :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function INT4(A)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) :: A</td>
</tr>
</tbody>
</table>

end

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT8</td>
<td>Convert to INTEGER(8) type.</td>
</tr>
</tbody>
</table>

Class. generic elemental nonstandard function

Summary.

generic INT8(A)

    INTEGER(8) function INT8(A)
    INTEGER(1) :: A
    INTEGER(8) function INT8(A)
    INTEGER(2) :: A
    INTEGER(8) function INT8(A)
    INTEGER(4) :: A
    INTEGER(8) function INT8(A)
    INTEGER(8) :: A
    INTEGER(8) function INT8(A)
    REAL :: A
    INTEGER(8) function INT8(A)
    DOUBLE PRECISION :: A
    INTEGER(8) function INT8(A)
    REAL(16) :: A
    INTEGER(8) function INT8(A)
    COMPLEX :: A
    INTEGER(8) function INT8(A)
    DOUBLE COMPLEX :: A
    INTEGER(8) function KIDINT(A)
    DOUBLE PRECISION :: A
    INTEGER(8) function INT8(A)
    COMPLEX(16) :: A
    INTEGER(8) function KIDINT(A)
    DOUBLE PRECISION :: A

end

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INUM</strong></td>
<td>Convert character to INTEGER(2) type. <strong>Class.</strong> specific elemental nonstandard function <strong>Summary.</strong> INTEGER(2) function INUM(I)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER :: I</td>
</tr>
<tr>
<td></td>
<td>NOTE: The ASCII characters in the CHARACTER expression must be numeric characters, and the result must fit in an INTEGER(2) number.</td>
</tr>
<tr>
<td><strong>IOR</strong></td>
<td>Logical OR. <strong>Class.</strong> generic elemental function <strong>Summary.</strong> generic IOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function BIOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: I,J</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function HIOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: I,J</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function IIOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: I,J</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function JIOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: I,J</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function KIOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: I,J</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td><strong>IQNINT</strong></td>
<td>see NINT</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISHFT</td>
<td>Logical shift. Class. generic elemental function Summary. generic ISHFT(I,SHIFT)</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER(1) function</td>
<td></td>
</tr>
<tr>
<td>BSHFT(I,SHIFT)</td>
<td>INTEGER(1) ::I,SHIFT</td>
</tr>
<tr>
<td>INTEGER(2) function</td>
<td></td>
</tr>
<tr>
<td>HSHFT(I,SHIFT)</td>
<td>INTEGER(2) ::I,SHIFT</td>
</tr>
<tr>
<td>INTEGER(2) function</td>
<td></td>
</tr>
<tr>
<td>IISHFT(I,SHIFT)</td>
<td>INTEGER(2) ::I,SHIFT</td>
</tr>
<tr>
<td>INTEGER(4) function</td>
<td></td>
</tr>
<tr>
<td>JISHFT(I,SHIFT)</td>
<td>INTEGER(4) ::I,SHIFT</td>
</tr>
<tr>
<td>INTEGER(8) function</td>
<td></td>
</tr>
<tr>
<td>KISHFT(I,SHIFT)</td>
<td>INTEGER(8) ::I,SHIFT</td>
</tr>
</tbody>
</table>

end
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISHFTC</td>
<td>Circular shift of the rightmost bits.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic ISHFTC(I,SHIFT,SIZE)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function</td>
</tr>
<tr>
<td></td>
<td>BSHIFT(I,SHIFT,SIZE)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1),OPTIONAL :: SIZE</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>HSHIFT(I,SHIFT,SIZE)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2),OPTIONAL :: SIZE</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>IISHFTC(I,SHIFT,SIZE)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2),OPTIONAL :: SIZE</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>JISHFTC(I,SHIFT,SIZE)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4),OPTIONAL :: SIZE</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>KISHFTC(I,SHIFT,SIZE)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8),OPTIONAL :: SIZE</td>
</tr>
</tbody>
</table>

end
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ISIGN**           | Absolute value of A times the sign of B.  
|                     | See also SIGN.  
| **Class.**          | generic nonstandard function  
| **Summary.**        | generic ISIGN(A,B)  
|                     | INTEGER(1) function BSIGN(A,B)  
|                     | INTEGER(1) ::A,B  
|                     | INTEGER(2) function IISIGN(A,B)  
|                     | INTEGER(2) ::A,B  
|                     | INTEGER(4) function JISIGN(A,B)  
|                     | INTEGER(4) ::A,B  
|                     | INTEGER(8) function KISIGN(A,B)  
|                     | INTEGER(8) ::A,B  
|                     | end |
| **ISNAN**           | Determine if a value is NaN.  
| **Class.**          | generic elemental nonstandard function  
| **Summary.**        | generic ISNAN(X)  
|                     | LOGICAL function ISNAN(X)  
|                     | REAL ::X  
|                     | LOGICAL function ISNAN(X)  
|                     | DOUBLE PRECISION ::X  
|                     | LOGICAL function ISNAN(X)  
|                     | end |

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| IXOR                | Exclusive OR. Class. generic elemental nonstandard function Summary. generic IXOR(I,J)  
  INTEGER(1) function BIXOR(I,J)  
  INTEGER(1) ::I,J  
  INTEGER(2) function HIXOR(I,J)  
  INTEGER(2) ::I,J  
  INTEGER(2) function IIIXOR(I,J)  
  INTEGER(2) ::I,J  
  INTEGER(4) function JIXOR(I,J)  
  INTEGER(4) ::I,J  
  INTEGER(8) function IXOR(I,J)  
  INTEGER(8) ::I,J  
  end |
| JIABS               | see IABS |
| JIAND               | see IAND |
| JIBCLR              | see IBCLR |
| JIBITS              | see IBITS |
| JIBSET              | see IBSET |
| JIDIM               | see IDIM |
| JIDINT              | see IDINT |
| JIDNNT              | see NINT |
| JIEOR               | see IEOR |
| JIFIX               | see INT |
| JINT                | see INT |
| JIOR                | see IOR |
| JIQINT              | see IQINT |
| JIQNNT              | see NINT |

continued
Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JISHFT</td>
<td>see ISHFT</td>
</tr>
<tr>
<td>JISHFTC</td>
<td>see ISHFTC</td>
</tr>
<tr>
<td>JISIGN</td>
<td>see SIGN</td>
</tr>
<tr>
<td>JIXOR</td>
<td>see IXOR</td>
</tr>
<tr>
<td>JMAX0</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>JMAX1</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>JMIN0</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>JMIN1</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>JMOD</td>
<td>see “MOD(A, P)”</td>
</tr>
<tr>
<td>JNINT</td>
<td>see NINT</td>
</tr>
<tr>
<td>JNOT</td>
<td>see NOT</td>
</tr>
<tr>
<td>JNUM</td>
<td>Convert character to integer type.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> specific elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong> INTEGER(4) function JNUM(I)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER :: I</td>
</tr>
<tr>
<td></td>
<td>NOTE: The characters in the ASCII CHARACTER expression must be numeric, and the result must fit in an INTEGER(4) number.</td>
</tr>
<tr>
<td>KIABS</td>
<td>see IABS</td>
</tr>
<tr>
<td>KIAND</td>
<td>see IAND</td>
</tr>
<tr>
<td>KIBCLR</td>
<td>see IBCLR</td>
</tr>
<tr>
<td>KIBITS</td>
<td>see IBITS</td>
</tr>
<tr>
<td>KIBSET</td>
<td>see IBSET</td>
</tr>
<tr>
<td>KIDIM</td>
<td>see IDIM</td>
</tr>
<tr>
<td>KIDINT</td>
<td>see IDINT</td>
</tr>
<tr>
<td>KIDNNT</td>
<td>see NINT</td>
</tr>
<tr>
<td>KIEOR</td>
<td>see IEOR</td>
</tr>
<tr>
<td>KIFIX</td>
<td>see INT</td>
</tr>
</tbody>
</table>

continued
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KIND</td>
<td>Return the kind type parameter of the argument.</td>
</tr>
<tr>
<td>Class. generic inquiry function</td>
<td></td>
</tr>
<tr>
<td>Summary. generic KIND(X)</td>
<td></td>
</tr>
</tbody>
</table>

```fortran
INTEGER function KIND(X)
    INTEGER(1) :: X
    INTEGER function KIND(X)
    INTEGER(2) :: X
    INTEGER function KIND(X)
    INTEGER(4) :: X
    INTEGER function KIND(X)
    INTEGER(8) :: X
    INTEGER function KIND(X)
    REAL(4) :: X
    INTEGER function KIND(X)
    REAL(8) :: X
    INTEGER function KIND(X)
    REAL(16) :: X
    INTEGER function KIND(X)
    COMPLEX(4) :: X
    INTEGER function KIND(X)
    COMPLEX(8) :: X
    INTEGER function KIND(X)
    COMPLEX(16) :: X
    INTEGER function KIND(X)
    LOGICAL(1) :: X
    INTEGER function KIND(X)
    LOGICAL(2) :: X
    INTEGER function KIND(X)
    LOGICAL(4) :: X
    INTEGER function KIND(X)
    LOGICAL(8) :: X
end
```

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KINT</td>
<td>see INT</td>
</tr>
<tr>
<td>KIOR</td>
<td>see IOR</td>
</tr>
<tr>
<td>KIQINT</td>
<td>see IQINT</td>
</tr>
<tr>
<td>KIQNNT</td>
<td>see NINT</td>
</tr>
<tr>
<td>KISHFT</td>
<td>see ISHFT</td>
</tr>
<tr>
<td>KISHFTC</td>
<td>see ISHFTC</td>
</tr>
<tr>
<td>KISIGN</td>
<td>see SIGN</td>
</tr>
<tr>
<td>KMAX0</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>KMAX1</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>KMIN0</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>KMIN1</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>KMOD</td>
<td>see “MOD(A, P)”</td>
</tr>
<tr>
<td>KNINT</td>
<td>see NINT</td>
</tr>
<tr>
<td>KNOT</td>
<td>see NOT</td>
</tr>
</tbody>
</table>
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBOUND</td>
<td>Return lower bounds of an array.</td>
</tr>
</tbody>
</table>

**Class.** generic inquiry function

**Summary.**

generic LBOUND(ARRAY,DIM)

!ARRAY must be array-valued, DIM must be scalar

```fortran
INTEGER function LBOUND(ARRAY,DIM)
  LOGICAL(1) :: ARRAY;
  INTEGER,OPTIONAL :: DIM
  INTEGER function LBOUND(ARRAY,DIM)
  LOGICAL(2) :: ARRAY;
  INTEGER,OPTIONAL :: DIM
  INTEGER function LBOUND(ARRAY,DIM)
  LOGICAL(4) :: ARRAY;
  INTEGER,OPTIONAL :: DIM
  INTEGER function LBOUND(ARRAY,DIM)
  LOGICAL(8) :: ARRAY;
  INTEGER,OPTIONAL :: DIM
  INTEGER function LBOUND(ARRAY,DIM)
```

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBOUND (continued)</td>
<td>(continued)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>LBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>LBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>LBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>LBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>REAL(4) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>LBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>REAL(8) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>LBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(4) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>(continued)</td>
</tr>
</tbody>
</table>

---

*Intrinsic Procedures*
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| LBOUND (continued)  | INTEGER function LBOUND(ARRAY,DIM)  
COMPLEX(8) ::ARRAY;  
INTEGER,OPTIONAL ::DIM  
INTEGER function LBOUND(ARRAY,DIM)  
COMPLEX(16) ::ARRAY;  
INTEGER,OPTIONAL ::DIM  
INTEGER function LBOUND(ARRAY,DIM)  
CHARACTER ::ARRAY;  
INTEGER,OPTIONAL ::DIM  
INTEGER function LBOUND(ARRAY,DIM)  
DERIVED_TYPE ::ARRAY;  
INTEGER, OPTIONAL ::DIM |
| LEN                 | Return the length of a character argument. |
|                     | **Class.** generic inquiry function |
|                     | **Summary.** |
|                     | generic LEN(STRING)  
INTEGER function LEN(STRING)  
CHARACTER::STRING |
| LEN_TRIM            | Length of a character string not including trailing blanks. |
|                     | **Class.** generic inquiry function |
|                     | **Summary.** |
|                     | generic LEN_TRIM(STRING)  
INTEGER function LEN_TRIM(STRING)  
CHARACTER::STRING |

**Table 1-3  Generic and Specific Intrinsic Procedures (continued)**
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LGE</strong></td>
<td>Lexical greater than or equal to comparison for strings.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td></td>
</tr>
<tr>
<td>generic LGE(STRING_A,STRING_B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOGICAL function</td>
</tr>
<tr>
<td></td>
<td>LGE(STRING_A,STRING_B)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td></td>
<td>::STRING_A,STRING_B</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td><strong>LGT</strong></td>
<td>Lexical greater than comparison for strings.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td></td>
</tr>
<tr>
<td>generic LGT(STRING_A,STRING_B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOGICAL function</td>
</tr>
<tr>
<td></td>
<td>LGT(STRING_A,STRING_B)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td></td>
<td>::STRING_A,STRING_B</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td><strong>LLE</strong></td>
<td>Lexical less than or equal to comparison for strings.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td></td>
</tr>
<tr>
<td>generic LLE(STRING_A,STRING_B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOGICAL function</td>
</tr>
<tr>
<td></td>
<td>LLE(STRING_A,STRING_B)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td></td>
<td>::STRING_A,STRING_B</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| LLT                 | Lexical less than comparison for strings.  
  **Class.** generic elemental function  
  **Summary.**  
  generic LLT(STRING_A,STRING_B)  
  LOGICAL function  
  LLT(STRING_A,STRING_B)  
  CHARACTER  
  ::STRING_A,STRING_B  
  end |
| LOC                 | Return the address of the argument.  
  **Class.** generic inquiry nonstandard function  
  **Summary.**  
  INTEGER function LOC(X)  
  ! X may be any one of the following:  
  INTEGER(1) ::X  
  INTEGER(2) ::X  
  INTEGER(4) ::X  
  INTEGER(8) ::X  
  REAL ::X  
  DOUBLE PRECISION ::X  
  REAL(16)::X  
  COMPLEX ::X  
  DOUBLE COMPLEX ::X  
  COMPLEX(16)::X  
  LOGICAL(1) ::X  
  LOGICAL(2) ::X  
  LOGICAL(4) ::X  
  LOGICAL(8) ::X |

Table 1-3  Generic and Specific Intrinsic Procedures  (continued)
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| LOC (continued)     | CHARACTER ::X  
                       DERIVED TYPE ::X  
                       **Note:** For Itanium®-based applications, the returned type of the LOC function is INTEGER*8. |
| LOG                 | Natural logarithm. |
| **Class.** generic elemental function |
| **Summary.** |
| generic LOG(X)     | 
                       REAL function ALOG(X)  
                       REAL ::X  
                       DOUBLE PRECISION function  
                       DLOG(X)  
                       DOUBLE PRECISION ::X  
                       REAL(16) function QLOG(X)  
                       REAL(16) :: X  
                       COMPLEX function CLOG(X)  
                       COMPLEX ::X  
                       DOUBLE COMPLEX function  
                       CDLOG(X)  
                       DOUBLE COMPLEX ::X  
                       DOUBLE COMPLEX function  
                       ZLOG(X)  
                       DOUBLE COMPLEX ::X  
                       COMPLEX(16) CQLOG(X)  
                       COMPLEX(16) :: X  
                       end |

**continued**
Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG10</td>
<td>Common logarithm.</td>
</tr>
<tr>
<td></td>
<td>Class. generic elemental function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic LOG10(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function ALOG10(X)</td>
</tr>
<tr>
<td></td>
<td>REAL :: X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DLOG10(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QLOG10(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGICAL</td>
<td>Convert to logical type.</td>
</tr>
</tbody>
</table>

**Class.** generic elemental function  
**Summary.**

generic LOGICAL(L,KIND)

LOGICAL(KIND) function
LOGICAL(L,KIND)
  LOGICAL(1) :: L
  INTEGER, OPTIONAL :: KIND
LOGICAL(KIND) function
LOGICAL(L,KIND)
  LOGICAL(2) :: L
  INTEGER, OPTIONAL :: KIND
LOGICAL(KIND) function
LOGICAL(L,KIND)
  LOGICAL(4) :: L
  INTEGER, OPTIONAL :: KIND
LOGICAL(KIND) function
LOGICAL(L,KIND)
  LOGICAL(8) :: L
  INTEGER, OPTIONAL :: KIND
LOGICAL function LOGICAL(L)
  LOGICAL(1) :: L
LOGICAL function LOGICAL(L)
  LOGICAL(2) :: L
LOGICAL function LOGICAL(L)
  LOGICAL(4) :: L
LOGICAL function LOGICAL(L)
  LOGICAL(8) :: L

end
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| LSHFT               | Left shift.
|                     | **Class.** generic elemental nonstandard function |
|                     | **Summary.** generic LSHFT(I,SHIFT) |
|                     | INTEGER(1) function |
|                     | LSHFT(I,SHIFT) |
|                     | INTEGER(1) ::I,SHIFT |
|                     | INTEGER(2) function |
|                     | LSHFT(I,SHIFT) |
|                     | INTEGER(2) ::I,SHIFT |
|                     | INTEGER(4) function |
|                     | LSHFT(I,SHIFT) |
|                     | INTEGER(4) ::I,SHIFT |
|                     | INTEGER(8) function |
|                     | LSHFT(I,SHIFT) |
|                     | INTEGER(8) ::I,SHIFT |
|                     | end |
|                     | **Note:** SHIFT must be a positive integer expression of the same KIND as I.
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSHIFT</td>
<td>Left shift.</td>
</tr>
<tr>
<td></td>
<td>Class. generic elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic LSHIFT(I,SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function</td>
</tr>
<tr>
<td></td>
<td>LSHIFT(I,SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>LSHIFT(I,SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>LSHIFT(I,SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>LSHIFT(I,SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::I,SHIFT</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: SHIFT must be a positive integer expression of the same KIND as I.</td>
</tr>
</tbody>
</table>

| MALLOC(I)           | The starting address for the block of memory. |
|                     | Class. elemental nonstandard function |

Table 1-3  **Generic and Specific Intrinsic Procedures**  (continued)
MATMUL
Matrix multiply.

Class. generic transformational function

Summary.
generic MATMUL(MATRIX_A, MATRIX_B)

Notes.
MATRIX_A must be of numeric type or of logical type and must
be array-valued and of rank one or two.
MATRIX_B must be of numeric type if MATRIX_A is of numeric
type, or of logical type if MATRIX_A is of logical type. If
MATRIX_A has rank one, MATRIX_B must have rank two; if
MATRIX_A has rank two, MATRIX_B must have rank one.

If the arguments are logical, the result is of type logical.
If the arguments are of numeric type, the result has numeric type.

MAX
Maximum value.

Class. generic elemental function

Summary.
generic MAX(A1, A2, ...)

REAL function
AIMAX0(A1, A2, ...)
INTEGER(2) :: A1, A2, ...

REAL function
AJMAX0(A1, A2, ...)
INTEGER(4) :: A1, A2, ...

REAL function
AKMAX0(A1, A2, ...)
INTEGER(8) :: A1, A2, ...

REAL function AMAX0(A1, A2, ...)
INTEGER :: A1, A2, ...

INTEGER(2) function
IMAX1(A1, A2, ...)
REAL :: A1, A2, ...

INTEGER(4) function
JMAX1(A1, A2, ...)

continued
### Intrinsic Procedures

- **MAX** (continued)
  - REAL :: A1, A2, ...
  - INTEGER function
  - MAX1(A1, A2, ...)
    - REAL :: A1, A2, ...
    - INTEGER(8) function
  - KMAX1(A1, A2, ...)
    - REAL :: A1, A2, ...
    - INTEGER(1) function
  - MAX(A1, A2, ...)
    - INTEGER(1) :: A1, A2, ...
    - INTEGER(2) function
  - IMAX0(A1, A2, ...)
    - INTEGER(2) :: A1, A2, ...
    - INTEGER(4) function
  - JMAX0(A1, A2, ...)
    - INTEGER(4) :: A1, A2, ...
    - INTEGER function
  - MAX0(A1, A2, ...)
    - INTEGER :: A1, A2, ...
    - INTEGER(8) function
  - KMAX0(A1, A2, ...)
    - INTEGER(8) :: A1, A2, ...
    - REAL function AMAX1(A1, A2, ...)
    - REAL :: A1, A2, ...
    - DOUBLE PRECISION function
  - XMAX1(A1, A2, ...)
    - DOUBLE PRECISION
    - :: A1, A2, ...

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| MAX (continued)     | REAL :: A1, A2, ...
|                     | INTEGER function |
|                     | MAX1(A1, A2, ...) |
|                     | REAL :: A1, A2, ...
|                     | INTEGER(8) function |
|                     | KMAX1(A1, A2, ...) |
|                     | REAL :: A1, A2, ...
|                     | INTEGER(1) function |
|                     | MAX(A1, A2, ...) |
|                     | INTEGER(1) :: A1, A2, ...
|                     | INTEGER(2) function |
|                     | IMAX0(A1, A2, ...) |
|                     | INTEGER(2) :: A1, A2, ...
|                     | INTEGER(4) function |
|                     | JMAX0(A1, A2, ...) |
|                     | INTEGER(4) :: A1, A2, ...
|                     | INTEGER function |
|                     | MAX0(A1, A2, ...) |
|                     | INTEGER :: A1, A2, ...
|                     | INTEGER(8) function |
|                     | KMAX0(A1, A2, ...) |
|                     | INTEGER(8) :: A1, A2, ...
|                     | REAL function AMAX1(A1, A2, ...) |
|                     | REAL :: A1, A2, ...
|                     | DOUBLE PRECISION function |
|                     | XMAX1(A1, A2, ...) |
|                     | DOUBLE PRECISION |
|                     | :: A1, A2, ...

**Table 1-3** Generic and Specific Intrinsic Procedures (continued)
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX0</td>
<td>see “MAX(A1, A2, A3, ...)”</td>
</tr>
<tr>
<td>MAX1</td>
<td>see “MAX(A1, A2, A3, ...)”</td>
</tr>
<tr>
<td>MAXEXPONENT</td>
<td>Return the maximum exponent in the model representing numbers of the same type and kind type parameter as the argument.</td>
</tr>
</tbody>
</table>

**Class.** generic inquiry function

**Summary.**

```
generic MAXEXPONENT(X)
    INTEGER function MAXEXPONENT(X)
        REAL :: X
    INTEGER function MAXEXPONENT(X)
        DOUBLE PRECISION :: X
    INTEGER function MAXEXPONENT(X)
        REAL(16) :: X
end
```

continued
### Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXLOC</td>
<td>Returns location of the first element of an array having the maximum value of elements identified by MASK.</td>
</tr>
</tbody>
</table>

**Class.** generic transformational function  

**Summary.**

generic MAXLOC(ARRAY,DIM,MASK,KIND)

```fortran
INTEGER function MAXLOC(ARRAY,DIM,MASK,KIND)
INTEGER(1) :: ARRAY;
INTEGER,OPTIONAL :: DIM
LOGICAL,OPTIONAL :: MASK
INTEGER,OPTIONAL :: KIND

INTEGER function MAXLOC(ARRAY,DIM,MASK,KIND)
INTEGER(2) :: ARRAY;
INTEGER,OPTIONAL :: DIM
LOGICAL,OPTIONAL :: MASK
INTEGER,OPTIONAL :: KIND

INTEGER function MAXLOC(ARRAY,DIM,MASK,KIND)
INTEGER(4) :: ARRAY;
INTEGER,OPTIONAL :: DIM
LOGICAL,OPTIONAL :: MASK
INTEGER,OPTIONAL :: KIND

INTEGER function MAXLOC(ARRAY,DIM,MASK,KIND)
INTEGER(8) :: ARRAY;
INTEGER,OPTIONAL :: DIM
LOGICAL,OPTIONAL :: MASK
INTEGER,OPTIONAL :: KIND
```

continued
Intrinsic Procedure  Description

MAXLOC (continued)  INTEGER function
MAXLOC (ARRAY, DIM, MASK, KIND)
REAL(4) :: ARRAY;
INTEGER, OPTIONAL :: DIM
LOGICAL, OPTIONAL :: MASK
INTEGER, OPTIONAL :: KIND

INTEGER function
MAXLOC (ARRAY, DIM, MASK, KIND)
REAL(8) :: ARRAY;
INTEGER, OPTIONAL :: DIM
LOGICAL, OPTIONAL :: MASK
INTEGER, OPTIONAL :: KIND

INTEGER function
MAXLOC (ARRAY, DIM, MASK, KIND)
REAL(16) :: ARRAY;
INTEGER, OPTIONAL :: DIM
LOGICAL, OPTIONAL :: MASK
INTEGER, OPTIONAL :: KIND

INTEGER function
MAXLOC (ARRAY, DIM, MASK, KIND)
end
### Intrinsic Procedures

**MAXVAL**

Maximum value of elements of array along dimension DIM that correspond to `.TRUE.` elements of MASK.

**Class.** generic transformational function

**Summary.**

```
generic MAXVAL(ARRAY,MASK,DIM)

! ARRAY must be array-valued
INTEGER(1) function MAXVAL(ARRAY,DIM,MASK)
  INTEGER(1) :: ARRAY
  INTEGER,OPTIONAL :: DIM
  LOGICAL,OPTIONAL :: MASK

INTEGER(2) function MAXVAL(ARRAY,DIM,MASK)
  INTEGER(2) :: ARRAY
  INTEGER,OPTIONAL :: DIM
  LOGICAL,OPTIONAL :: MASK

INTEGER(4) function MAXVAL(ARRAY,DIM,MASK)
  INTEGER(4) :: ARRAY
  INTEGER,OPTIONAL :: DIM
  LOGICAL,OPTIONAL :: MASK

INTEGER(8) function MAXVAL(ARRAY,DIM,MASK)
  INTEGER(8) :: ARRAY
  INTEGER,OPTIONAL :: DIM
  LOGICAL,OPTIONAL :: MASK
```

---

**Table 1-3 Generic and Specific Intrinsic Procedures (continued)**

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXVAL</td>
<td>Maximum value of elements of array along dimension DIM that correspond to <code>.TRUE.</code> elements of MASK.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic transformational function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

Continued...
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXVAL (continued)</td>
<td>MAXVAL(ARRAY,DIM,MASK)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>REAL(4) function</td>
</tr>
<tr>
<td></td>
<td>MAXVAL(ARRAY,DIM,MASK)</td>
</tr>
<tr>
<td></td>
<td>REAL(4) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>REAL(8) function</td>
</tr>
<tr>
<td></td>
<td>MAXVAL(ARRAY,DIM,MASK)</td>
</tr>
<tr>
<td></td>
<td>REAL(8) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function</td>
</tr>
<tr>
<td></td>
<td>MAXVAL(ARRAY,DIM,MASK)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>MCLOCK</td>
<td>Return time accounting for a program.</td>
</tr>
<tr>
<td></td>
<td>Class. specific inquiry nonstandard function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>INTEGER function MCLOCK()</td>
</tr>
<tr>
<td>MERGE</td>
<td>Choose alternative value according to the value of a mask.</td>
</tr>
<tr>
<td></td>
<td>Class. generic elemental function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic MERGE(TSOURCE,FSOURCE,MASK)</td>
</tr>
<tr>
<td></td>
<td>TSOURCE may be of any type.</td>
</tr>
<tr>
<td></td>
<td>FSOURCE must be of the same type as TSOURCE.</td>
</tr>
<tr>
<td></td>
<td>MASK must be type logical.</td>
</tr>
</tbody>
</table>

continued
### Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| MIN                 | Minimum value.  
Class. generic elemental function  
Summary.  
generic MIN(A1, A2,...)  

    REAL function  
AIMIN0(A1, A2,...)  
    INTEGER(2) :: A1, A2,...  
    REAL function  
AJMIN0(A1, A2,...)  
    INTEGER(4) :: A1, A2,...  
    REAL function  
AKMIN0(A1, A2,...)  
    INTEGER(8) :: A1, A2,...  
    REAL function  
AMIN0(A1, A2,...)  
    INTEGER :: A1, A2,...  
    INTEGER(2) function  
IMIN1(A1, A2,...)  
    REAL :: A1, A2,...  
    INTEGER(4) function  
JMIN1(A1, A2,...)  
    REAL :: A1, A2,...  
    INTEGER function  
MIN1(A1, A2,...)  
    REAL :: A1, A2,...  
    INTEGER(8) function  
KMIN1(A1, A2,...)  
    REAL :: A1, A2,...  
    INTEGER(1) function  
MIN(A1, A2,...)  
    INTEGER(1) :: A1, A2,...  
    INTEGER(2) function  
IMIN0(A1, A2,...)  

continued
Table 1-3  **Generic and Specific Intrinsic Procedures**  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN (continued)</td>
<td>INTEGER(2) ::A1,A2,...</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>JMIN0(A1,A2,...)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::A1,A2,...</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>MIN0(A1,A2,...)</td>
</tr>
<tr>
<td></td>
<td>INTEGER ::A1,A2,...</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>KMIN0(A1,A2,...)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::A1,A2,...</td>
</tr>
<tr>
<td></td>
<td>REAL function AMIN1(A1,A2,...)</td>
</tr>
<tr>
<td></td>
<td>REAL ::A1,A2,...</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DMIN1(A1,A2,...)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::A1,A2,...</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function</td>
</tr>
<tr>
<td></td>
<td>QMIN1(A1,A2,...)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: A1,A2,...</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>MIN0</td>
<td>see “MIN(A1,A2,A3,...)”</td>
</tr>
<tr>
<td>MIN1</td>
<td>see “MIN(A1,A2,A3,...)”</td>
</tr>
</tbody>
</table>

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINEXPONENT</td>
<td>Return the minimum exponent in the model representing numbers of the same type and kind type parameter as the argument.</td>
</tr>
</tbody>
</table>

**Class.** generic inquiry function

**Summary.**

generic MINEXPONENT(X)

    INTEGER function MINEXPONENT(X)
    REAL :: X

    INTEGER function MINEXPONENT(X)
    DOUBLE PRECISION :: X

end
INTEL FORTRAN LIBRARIES REFERENCE

Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINLOC</td>
<td>Return location of the first element of an array having the minimum value of elements identified by MASK.</td>
</tr>
<tr>
<td>Class.</td>
<td>generic transformational function</td>
</tr>
<tr>
<td>Summary.</td>
<td>generic MINLOC(ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>! ARRAY must be array-valued</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>MINLOC (ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>MINLOC (ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1),OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>MINLOC (ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>MINLOC (ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>continued</td>
</tr>
<tr>
<td>Intrinsic Procedure</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MINLOC (continued)</td>
<td>INTEGER function MINLOC(ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>REAL(4) ::ARRAY INTEGER,OPTIONAL ::DIM LOGICAL,OPTIONAL ::MASK INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function MINLOC(ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>REAL(8) ::ARRAY INTEGER,OPTIONAL ::DIM LOGICAL,OPTIONAL ::MASK INTEGER,OPTIONAL ::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function MINLOC(ARRAY,DIM,MASK,KIND)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) ::ARRAY INTEGER,OPTIONAL ::DIM LOGICAL,OPTIONAL ::MASK INTEGER,OPTIONAL ::KIND</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINVAL</td>
<td>Minimum value of elements of array along dimension DIM that correspond to .TRUE. elements of MASK.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic transformational function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong> generic MINVAL(ARRAY, MASK, DIM)</td>
</tr>
<tr>
<td></td>
<td>! ARRAY must be array-valued</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function</td>
</tr>
<tr>
<td></td>
<td>MINVAL (ARRAY, DIM, MASK)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: DIM;</td>
</tr>
<tr>
<td></td>
<td>LOGICAL, OPTIONAL :: MASK</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>MINVAL (ARRAY, DIM, MASK)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: DIM;</td>
</tr>
<tr>
<td></td>
<td>LOGICAL, OPTIONAL :: MASK</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>MINVAL (ARRAY, DIM, MASK)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: DIM;</td>
</tr>
<tr>
<td></td>
<td>LOGICAL, OPTIONAL :: MASK</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>MINVAL (ARRAY, DIM, MASK)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: ARRAY</td>
</tr>
<tr>
<td></td>
<td>continued</td>
</tr>
</tbody>
</table>
### Intrinsic Procedures

#### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| MINVAL -(continued) | INTEGER, OPTIONAL ::DIM;  
LOGICAL, OPTIONAL ::MASK 
REAL function
MINVAL (ARRAY, DIM, MASK)  
REAL ::ARRAY  
INTEGER, OPTIONAL ::DIM;  
LOGICAL, OPTIONAL ::MASK 
REAL(8) function
MINVAL (ARRAY, DIM, MASK)  
DOUBLE PRECISION ::ARRAY  
INTEGER, OPTIONAL ::DIM;  
LOGICAL, OPTIONAL ::MASK 
REAL(16) function
MINVAL (ARRAY, DIM, MASK)  
REAL(16) ::ARRAY  
INTEGER, OPTIONAL ::DIM;  
LOGICAL, OPTIONAL ::MASK |

end  

---

continued
**Table 1-3  Generic and Specific Intrinsic Procedures (continued)**

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| MOD                 | Remainder function. **Class.** generic elemental function  **Summary.**
|                     | **generic MOD(A,P)** |

```
generic MOD(A,P)
  INTEGER(1) function BMOD(A,P)
  INTEGER(1) :: A, P
  LOGICAL(1) function BMOD(A,P)
  LOGICAL(1) :: A, P
  INTEGER(2) function HMOD(A,P)
  INTEGER(2) :: A, P
  INTEGER(2) function IMOD(A,P)
  INTEGER(2) :: A, P
  INTEGER(4) function JMOD(A,P)
  INTEGER(4) :: A, P
  INTEGER(8) function KMOD(A,P)
  INTEGER(8) :: A, P
  REAL function AMOD(A,P)
  REAL :: A, P
  DOUBLE PRECISION function DMOD(A,P)
  DOUBLE PRECISION :: A, P
  REAL(16) function QMOD(A,P)
  REAL(16) :: A, P
end
```

continued
Table 1-3  **Generic and Specific Intrinsic Procedures**  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| MODULO              | Modulo function.  

**Class.** generic elemental function

**Summary.**

generic MODULO(A,P)

INTEGER(1) function MODULO(A,P)

INTEGER(1) ::A,P

INTEGER(2) function MODULO(A,P)

INTEGER(2) ::A,P

INTEGER(4) function MODULO(A,P)

INTEGER(4) ::A,P

INTEGER(8) function MODULO(A,P)

INTEGER(8) ::A,P

REAL function MODULO(A,P)

REAL ::A,P

DOUBLE PRECISION function MODULO(A,P)

DOUBLE PRECISION ::A,P

end

continued
Table 1-3 **Generic and Specific Intrinsic Procedures** (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVBITS</td>
<td>Copy a sequence of bits from one data object to another.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental subroutine</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic MVBITS(FROM,FROMPOS,LEN,TO,TOPOS)</td>
</tr>
<tr>
<td></td>
<td>subroutine BMVBITS(FROM,FROMPOS,LEN,TO,TOPOS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1):: FROM,TO</td>
</tr>
<tr>
<td></td>
<td>INTEGER :: FROMPOS, TOPOS, LEN</td>
</tr>
<tr>
<td></td>
<td>subroutine HMVBITS(FROM,FROMPOS,LEN,TO,TOPOS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2):: FROM,TO</td>
</tr>
<tr>
<td></td>
<td>INTEGER :: FROMPOS, TOPOS, LEN</td>
</tr>
<tr>
<td></td>
<td>subroutine MVBITS(FROM,FROMPOS,LEN,TO,TOPOS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4):: FROM,TO</td>
</tr>
<tr>
<td></td>
<td>INTEGER :: FROMPOS, TOPOS, LEN</td>
</tr>
<tr>
<td></td>
<td>subroutine MVBITS(FROM,FROMPOS,LEN,TO,TOPOS)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8):: FROM,TO</td>
</tr>
<tr>
<td></td>
<td>INTEGER :: FROMPOS, TOPOS, LEN</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

continued
### Intrinsic Procedures

#### NEAREST

Return the nearest different machine representable number in a given direction.

**Class.** generic elemental function

**Summary.**

generic NEAREST(X,S)

```plaintext
REAL function NEAREST(X,S)
REAL :: X, S
DOUBLE PRECISION function NEAREST(X,S)
DOUBLE PRECISION :: X, S
REAL(16) function NEAREST(X,S)
REAL(16) :: X, S
end
```

#### NINT

Nearest integer.

**Class.** generic elemental function

**Summary.**

generic NINT(A,KIND)

```plaintext
INTEGER(KIND) function NINT(A,KIND)
REAL :: A;
INTEGER, OPTIONAL :: KIND
INTEGER(KIND) function NINT(A,KIND)
DOUBLE PRECISION :: A;
```

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEAREST</td>
<td>Return the nearest different machine representable number in a given direction.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td>generic NEAREST(X,S)</td>
<td>REAL function NEAREST(X,S)</td>
</tr>
<tr>
<td></td>
<td>REAL :: X, S</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function NEAREST(X,S)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X, S</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function NEAREST(X,S)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X, S</td>
</tr>
<tr>
<td>NINT</td>
<td>Nearest integer.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong></td>
</tr>
<tr>
<td>generic NINT(A,KIND)</td>
<td>INTEGER(KIND) function NINT(A,KIND)</td>
</tr>
<tr>
<td></td>
<td>REAL :: A;</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER(KIND) function NINT(A,KIND)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: A;</td>
</tr>
</tbody>
</table>
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NINT (continued)</td>
<td>Nearest integer.</td>
</tr>
<tr>
<td></td>
<td>Class: generic elemental function</td>
</tr>
<tr>
<td></td>
<td>Summary:</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL::KIND</td>
</tr>
<tr>
<td></td>
<td>INTEGER function IDNINT(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION::A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function ININT(A)</td>
</tr>
<tr>
<td></td>
<td>REAL A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function NINT(A)</td>
</tr>
<tr>
<td></td>
<td>REAL :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function IIDNNT(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION::A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function IIQNNT(A)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function JNINT(A)</td>
</tr>
<tr>
<td></td>
<td>REAL :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function JIDNNT(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION::A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function IQNINT(A)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function KNINT(A)</td>
</tr>
<tr>
<td></td>
<td>REAL :: A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function KIDNNT(A)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION::A</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function KIQNNT(A)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) A</td>
</tr>
</tbody>
</table>

end continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| NOT                 | Bitwise complement.  
  **Class.** generic elemental function  
  **Summary.**  
  generic NOT(I)  
  
  INTEGER(1) function BNOT(I)  
  INTEGER(1) ::I

  INTEGER(2) function HNOT(I)  
  INTEGER(2) ::I

  INTEGER(2) function INOT(I)  
  INTEGER(2) ::I

  INTEGER(4) function JNOT(I)  
  INTEGER(4) ::I

  INTEGER(8) function KNOT(I)  
  INTEGER(8) ::I |

| OR                  | Bitwise logical OR.  
  **Class.** generic elemental function  
  **Summary.**  
  generic OR(I,J)  
  
  INTEGER(1) function OR(I,J)  
  INTEGER(1) ::I,J

  INTEGER(2) function OR(I,J)  
  INTEGER(2) ::I,J

  INTEGER(4) function OR(I,J)  
  INTEGER(4) ::I,J

  INTEGER(8) function OR(I,J)  
  INTEGER(8) ::I,J |

**continued**
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PACK</strong></td>
<td>Pack an array into an array of rank one under control of a mask.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic transformational function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic PACK(ARRAY, MASK, VECTOR)</td>
</tr>
<tr>
<td><strong>Notes.</strong></td>
<td>ARRAY may be of any type; it must be array-valued.</td>
</tr>
<tr>
<td></td>
<td>MASK must be of type logical and must be conformable with ARRAY.</td>
</tr>
<tr>
<td></td>
<td>VECTOR is optional. It must be of same type as ARRAY and must be scalar.</td>
</tr>
<tr>
<td></td>
<td>The result is an array of rank one with the same type as ARRAY.</td>
</tr>
</tbody>
</table>

| **PRECISION**       | Return the decimal precision in the model representing real numbers with the same kind type parameter as the argument. |
| **Class.**          | generic inquiry function |
| **Summary.**        | generic PRECISION(X) |
|                     | INTEGER function PRECISION(X) |
|                     | REAL :: X |
|                     | INTEGER function PRECISION(X) |
|                     | DOUBLE PRECISION :: X |
|                     | INTEGER function PRECISION(X) |
|                     | REAL(16) :: X |
|                     | end |

---

Table 1-3  Generic and Specific Intrinsic Procedures  (continued)
### Intrinsic Procedures

#### MM_PREFETCH

Prefetch the cache line containing $R$ into the cache. Integer literal $I$ selects the type of prefetch.

**Class.** nonstandard function

**Summary.**

```fortran
SUBROUTINE MM_PREFETCH(R, I)
  REAL R
  INTEGER I

  I=0 ==> PREFETCH
  I=1 ==> PREFETCHTNT1
  I=2 ==> PREFETCHTNT2
  I=3 ==> PREFETCHTNTA
```

#### PRESENT

Determine whether an optional argument is present.

**Class.** generic inquiry function

**Summary.**

generic PRESENT(A)

**Notes.**

A may be of any type. It must be an optional argument of the procedure in which PRESENT function reference appears. The result has type LOGICAL.

---

Table 1-3  **Generic and Specific Intrinsic Procedures**  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM_PREFETCH</td>
<td>Prefetch the cache line containing $R$ into the cache. Integer literal $I$ selects the type of prefetch.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>nonstandard function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>SUBROUTINE MM_PREFETCH(R, I)</td>
</tr>
<tr>
<td></td>
<td>REAL R</td>
</tr>
<tr>
<td></td>
<td>INTEGER I</td>
</tr>
<tr>
<td></td>
<td>I=0 ==&gt; PREFETCH</td>
</tr>
<tr>
<td></td>
<td>I=1 ==&gt; PREFETCHTNT1</td>
</tr>
<tr>
<td></td>
<td>I=2 ==&gt; PREFETCHTNT2</td>
</tr>
<tr>
<td></td>
<td>I=3 ==&gt; PREFETCHTNTA</td>
</tr>
</tbody>
</table>

| PRESENT             | Determine whether an optional argument is present. |
| **Class.**          | generic inquiry function |
| **Summary.**        | generic PRESENT(A) |
| **Notes.**          | A may be of any type. It must be an optional argument of the procedure in which PRESENT function reference appears. The result has type LOGICAL. |
Table 1-3    Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT</td>
<td>Product of all elements of an array along dimension DIM corresponding to the .TRUE. elements of MASK.</td>
</tr>
</tbody>
</table>

**Class.** generic transformational function

**Summary.**

generic PRODUCT(ARRAY,DIM,MASK)

!ARRAY must be array-valued, DIM must be scalar

INTEGER(1) function
PRODUCT(ARRAY,DIM,MASK)

INTEGER(1) ::ARRAY
INTEGER,OPTIONAL ::DIM;
LOGICAL,OPTIONAL ::MASK

INTEGER(2) function
PRODUCT(ARRAY,DIM,MASK)

INTEGER(2) ::ARRAY
INTEGER,OPTIONAL ::DIM;
LOGICAL,OPTIONAL ::MASK

INTEGER(4) function
PRODUCT(ARRAY,DIM,MASK)

INTEGER(4) ::ARRAY
INTEGER,OPTIONAL ::DIM;
LOGICAL,OPTIONAL ::MASK

INTEGER(8) function
PRODUCT(ARRAY,DIM,MASK)

INTEGER(8) ::ARRAY
INTEGER,OPTIONAL ::DIM;
LOGICAL,OPTIONAL ::MASK

REAL(4) function
PRODUCT(ARRAY,DIM,MASK)

REAL(4) ::ARRAY

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT (continued)</td>
<td>INTEGER, OPTIONAL :: DIM; LOGICAL, OPTIONAL :: MASK</td>
</tr>
<tr>
<td></td>
<td>REAL(8) function</td>
</tr>
<tr>
<td></td>
<td>PRODUCT (ARRAY, DIM, MASK)</td>
</tr>
<tr>
<td></td>
<td>REAL(8) :: ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: DIM; LOGICAL, OPTIONAL :: MASK</td>
</tr>
<tr>
<td></td>
<td>LOGICAL, OPTIONAL :: MASK</td>
</tr>
</tbody>
</table>

end

QABS see "ABS(A)"

QACOS see "ACOS(X)"

QACOSD see "ACOSD(X)"

QACOSH see "ACOSH(X)"

QASIN see "ASIN(X)"

QASIND see "ASIND(X)"

QASINH see "ASINH(X)"

QATAN see "ATAN(X)"

QATAN2 see "ATAN2(Y, X)"

QATAN2D see "ATAN2D(Y, X)"

QATAND see "ATAND(X)"

QATANH see "ATANH(X)"

QCOS see "COS(X)"

QCOSD see "COSD(X)"

QCOSH see "COSH(X)"

QDIM see "DIM(X, Y)"

QEXP see "EXP(X)"

QINT see "AINT(A, KIND)"

QLOG see "LOG(X)"

QLOG10 see "LOG10(X)"
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMAX1</td>
<td>see “MAX(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>QMIN1</td>
<td>see “MIN(A1, A2, A3,...)”</td>
</tr>
<tr>
<td>QMOD</td>
<td>see “MOD(A, P)”</td>
</tr>
<tr>
<td>QNINT</td>
<td>see “ANINT(A, KIND)”</td>
</tr>
<tr>
<td>QPROD</td>
<td>Quad precision double product.</td>
</tr>
<tr>
<td></td>
<td>Class. generic elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic QPROD(X, Y)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::X, Y</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>QSIGN</td>
<td>see “SIGN(A, B)”</td>
</tr>
<tr>
<td>QSIN</td>
<td>see “SIN(X)”</td>
</tr>
<tr>
<td>QSIND</td>
<td>see “SIND(X)”</td>
</tr>
<tr>
<td>QSINH</td>
<td>see “SINH(X)”</td>
</tr>
<tr>
<td>QSQRT</td>
<td>see “SQRT(X)”</td>
</tr>
<tr>
<td>QTAN</td>
<td>see “TAN(X)”</td>
</tr>
<tr>
<td>QTAND</td>
<td>see “TAND(X)”</td>
</tr>
<tr>
<td>QTANH</td>
<td>see “TANH(X)”</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **RADIX**           | Return the base of the model representing numbers of the same type and kind type parameter as the argument.  
  **Class.** generic inquiry function  
  **Summary.**  
  generic RADIX(X) |
|                     | INTEGER function RADIX(X)  
  INTEGER(1) ::X  
  INTEGER function RADIX(X)  
  INTEGER(2) ::X  
  INTEGER function RADIX(X)  
  INTEGER(4) ::X  
  INTEGER function RADIX(X)  
  INTEGER(8) ::X  
  INTEGER function RADIX(X)  
  REAL ::X  
  INTEGER function RADIX(X)  
  DOUBLE PRECISION ::X |
| **RANDOM_NUMBER**   | Generate pseudorandom number in the range of 0. to 1.  
  **Class.** generic subroutine  
  **Summary.**  
  generic RANDOM_NUMBER(HARVEST) |
|                     | subroutine  
  RANDOM_NUMBER(HARVEST)  
  REAL ::HARVEST  
  subroutine  
  RANDOM_NUMBER(HARVEST)  
  DOUBLE PRECISION ::HARVEST |

---

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANDOM_SEED</td>
<td>Restart or query the pseudorandom number generator used by RANDOM_NUMBER.</td>
</tr>
<tr>
<td></td>
<td>Class. generic subroutine</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic RANDOM_SEED(SIZE,PUT,GET)</td>
</tr>
<tr>
<td></td>
<td>! There must be exactly one or no arguments</td>
</tr>
<tr>
<td></td>
<td>! PUT and GET must be a rank-one and</td>
</tr>
<tr>
<td></td>
<td>! must be of sufficient size</td>
</tr>
<tr>
<td></td>
<td>subroutine</td>
</tr>
<tr>
<td></td>
<td>RANDOM_SEED(SIZE,PUT,GET)</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL</td>
</tr>
<tr>
<td></td>
<td>::SIZE,PUT,GET</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

continued
Intrinsic Procedures

Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANGE</td>
<td>Return the decimal exponent range in the model representing integer or real numbers with the same kind type parameter as the argument.</td>
</tr>
</tbody>
</table>
| **Class.** generic inquiry function **Summary.**
| generic RANGE(X)    | |
| INTEGER function RANGE(X) |
| INTEGER(1) ::X       |
| INTEGER function RANGE(X) |
| INTEGER(2) ::X       |
| INTEGER function RANGE(X) |
| INTEGER(4) ::X       |
| INTEGER function RANGE(X) |
| INTEGER(8) ::X       |
| INTEGER function RANGE(X) |
| REAL(4) ::X          |
| INTEGER function RANGE(X) |
| REAL(8) ::X          |
| end                  |

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>Convert to real type.</td>
</tr>
<tr>
<td>Class. generic elemental function</td>
<td></td>
</tr>
<tr>
<td>Summary.</td>
<td></td>
</tr>
<tr>
<td>generic REAL(A,KIND)</td>
<td></td>
</tr>
<tr>
<td>REAL function FLOAT(A)</td>
<td></td>
</tr>
<tr>
<td>INTEGER ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function REAL(A)</td>
<td></td>
</tr>
<tr>
<td>INTEGER(1) ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function FLOATI(A)</td>
<td></td>
</tr>
<tr>
<td>INTEGER(2) ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function FLOATJ(A)</td>
<td></td>
</tr>
<tr>
<td>INTEGER(4) ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function FLOATK(A)</td>
<td></td>
</tr>
<tr>
<td>INTEGER(8) ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function SNGL(A)</td>
<td></td>
</tr>
<tr>
<td>REAL ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function REAL(A)</td>
<td></td>
</tr>
<tr>
<td>REAL ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function REAL(A)</td>
<td></td>
</tr>
<tr>
<td>REAL ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function REAL(A)</td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function REAL(A)</td>
<td></td>
</tr>
<tr>
<td>COMPLEX ::A</td>
<td></td>
</tr>
<tr>
<td>REAL function REAL(A)</td>
<td></td>
</tr>
<tr>
<td>DOUBLE COMPLEX ::A</td>
<td></td>
</tr>
</tbody>
</table>

end
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| **REPEAT**          | Concatenate several copies of a string.  
Class. generic transformational function  
Summary.  

generic REPEAT(STRING,NCOPIES)  
    CHARACTER function  
    REPEAT(STRING,NCOPIES)  
    CHARACTER ::STRING  
    INTEGER ::NCOPIES  
end |
| **RESHAPE**         | Construct an array of a specified shape from the elements of a  
given array.  
Class. generic transformational function  
Summary.  
generic RESHAPE(SOURCE,SHAPE,PAD,ORDER)  
Notes.  
    SOURCE may be of any type. It  
    must be array-valued.  
    SHAPE must be of type integer,  
    rank one and constant size.  
    PAD is optional. It must have the  
    same type as SOURCE.  
    ORDER is optional. It must be of  
    type integer.  
    The result has the same type as  
    SOURCE. |
| **RNUM**            | Convert character to real type.  
Class. specific elemental nonstandard function  
Summary.  
    REAL function RNUM(I)  
    CHARACTER ::I  
continued |
### Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| RRSPACING           | Return the reciprocal of the relative spacing of model numbers near the argument value.  
**Class.** generic elemental function  
**Summary.**  
`generic RRSPACING(X)`  
```
REAL function RRSPACING(X)
    REAL :: X
    DOUBLE PRECISION function
    RRSPACING(X)
    DOUBLE PRECISION :: X
    REAL(16) RRSPACING(X)
    REAL(16) :: X
end`
| RSHIFT              | Bitwise right shift.  
**Class.** generic elemental nonstandard function  
**Summary.**  
`generic RSHIFT(I,SHIFT)`  
```
INTEGER(1) function
RSHIFT(I,SHIFT)
    INTEGER(1) :: I,SHIFT
    INTEGER(2) function
RSHIFT(I,SHIFT)
    INTEGER(2) :: I,SHIFT
    INTEGER(4) function
RSHIFT(I,SHIFT)
    INTEGER(4) :: I,SHIFT
    INTEGER(8) function
RSHIFT(I,SHIFT)
    INTEGER(8) :: I,SHIFT
end`

**Note:** `SHIFT` must be a positive integer of the same KIND as `I`.  
continued
Intrinsic Procedures

Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSHIFT</strong></td>
<td>Bitwise right shift.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental nonstandard function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic RSHIFT(I,SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function</td>
</tr>
<tr>
<td></td>
<td>RSHIFT (I, SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) :: I, SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function</td>
</tr>
<tr>
<td></td>
<td>RSHIFT (I, SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) :: I, SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function</td>
</tr>
<tr>
<td></td>
<td>RSHIFT (I, SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) :: I, SHIFT</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function</td>
</tr>
<tr>
<td></td>
<td>RSHIFT (I, SHIFT)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) :: I, SHIFT</td>
</tr>
<tr>
<td><strong>end</strong></td>
<td>Note: SHIFT must be a positive integer of the same KIND as I.</td>
</tr>
<tr>
<td><strong>SCALE</strong></td>
<td>Return (X \times (b^I)) where (b) is the base in the model</td>
</tr>
<tr>
<td></td>
<td>representation of (X).</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic SCALE(X, I)</td>
</tr>
<tr>
<td></td>
<td>REAL function SCALE(X, I)</td>
</tr>
<tr>
<td></td>
<td>REAL :: X; INTEGER :: I</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>SCALE(X, I)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X; INTEGER :: I</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function SCALE(X, I)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td></td>
<td>INTEGER :: I</td>
</tr>
<tr>
<td><strong>end</strong></td>
<td>continued</td>
</tr>
<tr>
<td>Intrinsic Procedure</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SCAN</td>
<td>Scan a string for any one of the characters in a set of characters.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic elemental function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong> generic SCAN(STRING,SET,BACK)</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>SCAN(STRING,SET,BACK)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER :: STRING, SET</td>
</tr>
<tr>
<td></td>
<td>LOGICAL, OPTIONAL :: BACK</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>SELECTED_INT_KIND</td>
<td>Return the value of the kind type parameter of an integer data type</td>
</tr>
<tr>
<td></td>
<td>that represents all integer values ( n ) with (-10^r &lt; n &lt; 10^r).</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic transformational function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong> generic SELECTED_INT_KIND(R)</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>SELECTED_INT_KIND(R)</td>
</tr>
<tr>
<td></td>
<td>INTEGER :: R</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>SELECTED_REAL_KIND</td>
<td>Return the value of the kind type parameter of a real data type with</td>
</tr>
<tr>
<td></td>
<td>decimal precision of at least ( P ) digits and a decimal exponent range</td>
</tr>
<tr>
<td></td>
<td>of at least ( R ).</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic transformational function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong> generic SELECTED_REAL_KIND(P,R)</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>SELECTED_REAL_KIND(P,R)</td>
</tr>
<tr>
<td></td>
<td>INTEGER, OPTIONAL :: P,R</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Table 1-3  **Generic and Specific Intrinsic Procedures**  (continued)

continued
### Intrinsic Procedures

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| SET_EXPONENT        | Return the model number whose fractional part is the fractional part of the model representation of \( X \) and whose exponent part is \( I \).  
**Class.** generic elemental function  
**Summary.**  
generic SET_EXPONENT(X,I)  

    REAL function  
    SET_EXPONENT(X,I)  
    REAL :: X; INTEGER :: I  

DOUBLE PRECISION function  
SET_EXPONENT(X,I)  

    DOUBLE PRECISION :: X; INTEGER :: I  

REAL(16) function  
SET_EXPONENT(X,I)  

    REAL(16) :: X  

INTEGER :: I  

end |
| SHAPE               | Return the shape of an array or a scalar.  
**Class.** generic inquiry function  
**Summary.**  
generic SHAPE(SOURCE)  

**Notes.**  
SOURCE may be of any type. The result is of type integer and is an array of rank one. |

---

Table 1-3 **Generic and Specific Intrinsic Procedures** (continued)
### Table 1-3 Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGN</td>
<td>Absolute value of $A$ times the sign of $B$. <strong>Class.</strong> generic elemental function <strong>Summary.</strong> [\text{generic \ SIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(1) \ function \ \text{BSIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(1) :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(2) \ function \ \text{HSIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(2) :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(2) \ function \ \text{IISIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(2) :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(4) \ function \ \text{ISIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(4) :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(4) \ function \ \text{JSIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(4) :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(8) \ function \ \text{KISIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{INTEGER}(8) :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{REAL} \ function \ \text{SIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{REAL} :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{DOUBLE PRECISION} \ function \ \text{DSIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{DOUBLE PRECISION} :: A,B]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{REAL(16)} \ function \ \text{QSIGN}(A,B)]</td>
</tr>
<tr>
<td></td>
<td>[\hspace{1cm}\text{REAL(16)} :: A,B]</td>
</tr>
</tbody>
</table>

end

continued
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>Sine function that accepts input in radians.</td>
</tr>
<tr>
<td><strong>Class.</strong> generic elemental function</td>
<td></td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic SIN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function SIN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td>DSIN(X)</td>
<td>DOUBLE PRECISION ::X</td>
</tr>
<tr>
<td></td>
<td>COMPLEX function CSIN(X)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX function</td>
</tr>
<tr>
<td>CDSIN(X)</td>
<td>DOUBLE COMPLEX ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX function</td>
</tr>
<tr>
<td>ZSIN(X)</td>
<td>DOUBLE COMPLEX ::X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QSIN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) X</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) function CQSIN(X)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) ::X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

**continued**
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIND</td>
<td>Sine function that accepts input in degrees.</td>
</tr>
<tr>
<td></td>
<td>Class. generic elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic SIND(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function SIND(X)</td>
</tr>
<tr>
<td></td>
<td>REAL :: X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DSIND(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td>SINH</td>
<td>Hyperbolic sine function.</td>
</tr>
<tr>
<td></td>
<td>Class. generic elemental function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic SINH(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function SINH(X)</td>
</tr>
<tr>
<td></td>
<td>REAL :: X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function</td>
</tr>
<tr>
<td></td>
<td>DSINH(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION :: X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QSINH(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Table 1-3 Generic and Specific Intrinsic Procedures (continued)
**Table 1-3 Generic and Specific Intrinsic Procedures** (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| SIZE                | Return the number of elements of an array or the extent along a specified dimension.  
  **Class.** generic non-standard inquiry function  
  **Summary.** generic SIZE(ARRAY,DIM)  
  **Notes.**  
  ARRAY may be of any type. It must be array-valued.  
  DIM is optional. It must be of type integer and must not be array-valued.  
  The result is of type integer. |
| SIZEOF              | Return the number of bytes of storage used by the argument.  
  **Class.** generic inquiry function  
  **Summary.** generic SIZEOF(A)  
  **Note.** A may be of any type. The result is of type integer. If A is a pointer, the function returns the size of what A points to rather than the size of A itself. |
| SNGL                | see REAL |
| SNGLQ               | see REAL |
| SPACING             | Return the absolute spacing of model numbers near the argument value.  
  **Class.** generic elemental function  
  **Summary.**  
  generic SPACING(X)  
  REAL function SPACING(X)  
  REAL :: X  
  DOUBLE PRECISION function SPACING(X)  
  DOUBLE PRECISION :: X  
  end |

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPREAD</td>
<td>Replicate an array by adding a dimension.</td>
</tr>
<tr>
<td></td>
<td><strong>Class.</strong> generic transformational function</td>
</tr>
<tr>
<td></td>
<td><strong>Summary.</strong> generic SPREAD(SOURCE,DIM,NCOPIES)</td>
</tr>
<tr>
<td></td>
<td><strong>Notes.</strong></td>
</tr>
<tr>
<td></td>
<td>SOURCE may be of any type.</td>
</tr>
<tr>
<td></td>
<td>DIM must be of type integer and must be scalar.</td>
</tr>
<tr>
<td></td>
<td>NCOPIES must be of type integer and must be scalar.</td>
</tr>
<tr>
<td></td>
<td>The result has the same type as SOURCE.</td>
</tr>
</tbody>
</table>

continued
### Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQRT</td>
<td>Square root.</td>
</tr>
<tr>
<td><strong>Class.</strong> generic elemental function</td>
<td></td>
</tr>
<tr>
<td><strong>Summary.</strong> generic SQRT(X)</td>
<td></td>
</tr>
</tbody>
</table>

```
    REAL function SQRT(X)
    REAL :: X
    DOUBLE PRECISION function DSQRT(X)
    DOUBLE PRECISION :: X
    COMPLEX function CSQRT(X)
    COMPLEX :: X
    DOUBLE COMPLEX function CDSQRT(X)
    DOUBLE COMPLEX :: X
    DOUBLE COMPLEX function ZSQRT(X)
    DOUBLE COMPLEX :: X
    REAL(16) function QSQRT(X)
    REAL(16) :: X
    COMPLEX(16) function CQSQR(X)
    COMPLEX(16) :: X
```

```
end
```
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>Sum all the elements of ARRAY along dimension DIM corresponding to all the .TRUE. elements of MASK.</td>
</tr>
</tbody>
</table>

Class. generic transformational function

Summary.

generic SUM(ARRAY, MASK, DIM)

! ARRAY must be array-valued
INTEGER(1) function SUM(ARRAY, DIM, MASK)
  INTEGER(1) :: ARRAY
  INTEGER, OPTIONAL :: DIM;
  LOGICAL, OPTIONAL :: MASK
INTEGER(2) function SUM(ARRAY, DIM, MASK)
  INTEGER(2) :: ARRAY
  INTEGER, OPTIONAL :: DIM;
  LOGICAL, OPTIONAL :: MASK
INTEGER(4) function SUM(ARRAY, DIM, MASK)
  INTEGER(4) :: ARRAY
  INTEGER, OPTIONAL :: DIM;
  LOGICAL, OPTIONAL :: MASK
INTEGER(8) function SUM(ARRAY, DIM, MASK)
  INTEGER(8) :: ARRAY
  INTEGER, OPTIONAL :: DIM;
  LOGICAL, OPTIONAL :: MASK
REAL(4) function SUM(ARRAY, DIM, MASK)
  REAL(4) :: ARRAY;
  INTEGER, OPTIONAL :: DIM;
  LOGICAL, OPTIONAL :: MASK
end

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM (continued)</td>
<td>REAL(8) function</td>
</tr>
<tr>
<td></td>
<td>SUM(ARRAY,DIM,MASK)</td>
</tr>
<tr>
<td></td>
<td>REAL(8) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM;</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function</td>
</tr>
<tr>
<td></td>
<td>SUM(ARRAY,DIM,MASK)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) ::ARRAY</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM;</td>
</tr>
<tr>
<td></td>
<td>LOGICAL,OPTIONAL ::MASK</td>
</tr>
<tr>
<td>SYSTEM_CLOCK</td>
<td>Return integer data from a real-time clock.</td>
</tr>
<tr>
<td>Class. generic subroutine</td>
<td></td>
</tr>
<tr>
<td>Summary.</td>
<td></td>
</tr>
<tr>
<td>generic</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_CLOCK(COUNT,COUNT_RATE,COUNT_MAX)</td>
<td>subroutine</td>
</tr>
<tr>
<td></td>
<td>SYSTEM_CLOCK(COUNT,COUNT_RATE,COUNT_MAX)</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::COUNT,COUNT_RATE,COUNT_MAX</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAN</strong></td>
<td>Tangent in radians.</td>
</tr>
<tr>
<td><strong>Class.</strong></td>
<td>generic elemental function</td>
</tr>
<tr>
<td><strong>Summary.</strong></td>
<td>generic TAN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL function TAN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION function DTAN(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE PRECISION ::X</td>
</tr>
<tr>
<td></td>
<td>COMPLEX function CTAN(X)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX ::X</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX function ZTAN(X)</td>
</tr>
<tr>
<td></td>
<td>DOUBLE COMPLEX ::X</td>
</tr>
<tr>
<td></td>
<td>REAL(16) function QTAN(X)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) :: X</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) function CQTAN(X)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) :: X</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

| **TAND**            | Tangent function that accepts input in degrees. |
| **Class.**          | generic elemental nonstandard function |
| **Summary.**        | generic TAND(X) |
|                     | REAL function TAND(X) |
|                     | REAL ::X |
|                     | DOUBLE PRECISION function DTAND(X) |
|                     | DOUBLE PRECISION ::X |
|                     | REAL(16) function QTAND(X) |
|                     | REAL(16) :: X |
|                     | COMPLEX(16) function CQTAN(X) |
|                     | COMPLEX(16) :: X |
|                     | end |

(continued)
Table 1-3  **Generic and Specific Intrinsic Procedures**  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| TANH                | Hyperbolic tangent.  
  **Class.** generic elemental function  
  **Summary.**  
  generic TANH(X)  
  
  REAL function TANH(X)  
  REAL ::X  
  DOUBLE PRECISION function  
  DTANH(X)  
  DOUBLE PRECISION ::X  
  REAL(16) function QTANH  
  REAL(16) QTANH |
| TINY                | Return the smallest positive number in the model representing numbers of the same type and kind type parameter as the argument.  
  **Class.** generic inquiry function  
  **Summary.**  
  generic TINY(X)  
  
  REAL function TINY(X)  
  REAL ::X  
  DOUBLE PRECISION function  
  TINY(X)  
  DOUBLE PRECISION ::X  
  REAL(16) function TINY(X)  
  REAL(16) :: X |

Table 1-3 (continued)
Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
</table>
| TRANSFER            | Return a result with a physical representation identical to that of SOURCE but interpreted with the type and type parameters of MOLD.  
  **Class.** generic transformational function  
  **Summary.**  
  generic TRANSFER(SOURCE, MOLD, SIZE)  
  **Notes.**  
  SOURCE may be of any type.  
  MOLD may be of any type.  
  SIZE is optional. It must be a scalar of type integer.  
  The result has the same type as MOLD. |
| TRANSPOSE           | Transpose an array of rank two.  
  **Class.** generic transformational function  
  **Summary.**  
  generic TRANSPOSE(MATRIX)  
  **Notes.**  
  MATRIX may be of any type. It must have rank two.  
  The result has the same type as MATRIX. |
| TRIM                | Return the argument with trailing blank characters removed.  
  **Class.** generic transformational function  
  **Summary.**  
  generic TRIM(STRING)  
  **Notes.**  
  CHARACTER function  
  TRIM(STRING)  
  CHARACTER :: STRING  
  end |

continued
<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBOUND</td>
<td>Returns upper bounds of an array.</td>
</tr>
</tbody>
</table>

**Class.** generic inquiry function

**Summary.**

```fortran
generic UBOUND(ARRAY,DIM)

!ARRAY must be array-valued, DIM must be scalar
INTEGER function UBOUND (ARRAY,DIM)
  LOGICAL(1) :: ARRAY;
  INTEGER, OPTIONAL :: DIM
  INTEGER function UBOUND (ARRAY,DIM)
  LOGICAL(2) :: ARRAY;
  INTEGER, OPTIONAL :: DIM
  INTEGER function UBOUND (ARRAY,DIM)
  LOGICAL(4) :: ARRAY;
  INTEGER, OPTIONAL :: DIM
  INTEGER function UBOUND (ARRAY,DIM)
  LOGICAL(8) :: ARRAY;
  INTEGER, OPTIONAL :: DIM
  INTEGER function UBOUND (ARRAY,DIM)
  INTEGER(1) :: ARRAY;
  INTEGER, OPTIONAL :: DIM
  INTEGER function UBOUND (ARRAY,DIM)
  INTEGER(2) :: ARRAY;
  INTEGER, OPTIONAL :: DIM
  INTEGER function UBOUND (ARRAY,DIM)
  INTEGER(4) :: ARRAY;
  INTEGER, OPTIONAL :: DIM
```

continued
Table 1-3  Generic and Specific Intrinsic Procedures  (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBOUND (continued)</td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>REAL(4) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>REAL(8) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>REAL(16) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(4) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(8) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>COMPLEX(16) ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
<tr>
<td></td>
<td>INTEGER function</td>
</tr>
<tr>
<td></td>
<td>UBOUND (ARRAY,DIM)</td>
</tr>
<tr>
<td></td>
<td>CHARACTER ::ARRAY;</td>
</tr>
<tr>
<td></td>
<td>INTEGER,OPTIONAL ::DIM</td>
</tr>
</tbody>
</table>

continued
**Table 1-3  Generic and Specific Intrinsic Procedures  (continued)**

<table>
<thead>
<tr>
<th>Intrinsic Procedure (continued)</th>
<th>Description</th>
</tr>
</thead>
</table>
| **UBOUND (continued)** | INTEGER function  
UBOUND(ARRAY,DIM)  
  
  **DERIVED_TYPE** ::ARRAY;  
  INTEGER,OPTIONAL ::DIM  
end |
| **UNPACK** | Unpack an array of rank one into an array under control of a mask.  
***Class.*** generic transformational function  
***Summary.***  
generic UNPACK(VECTOR,MASK,FIELD)  
***Notes.***  
  VECTOR may be of any type. It must have rank one.  
  MASK must have logical type. It must be array-valued.  
  FIELD must be the same type as VECTOR.  
  The result has the same type as VECTOR. |
| **VERIFY** | Verify that a set of a characters contains all the characters in a string by identifying the position of the first character that does not appear in a given set of characters.  
***Class.*** generic elemental function  
***Summary.***  
generic VERIFY(STRING,SET,BACK)  
  INTEGER function  
VERIFY(STRING,SET,BACK)  
CHARACTER ::STRING, SET  
LOGICAL,OPTIONAL ::BACK  
end |

*continued*
Table 1-3  Generic and Specific Intrinsic Procedures (continued)

<table>
<thead>
<tr>
<th>Intrinsic Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>Bitwise exclusive OR.</td>
</tr>
<tr>
<td></td>
<td>Class. general elemental nonstandard function</td>
</tr>
<tr>
<td></td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>generic XOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) function XOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(1) ::I,J</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) function XOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(2) ::I,J</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) function XOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(4) ::I,J</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) function XOR(I,J)</td>
</tr>
<tr>
<td></td>
<td>INTEGER(8) ::I,J</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>Note: this function can also be specified in all its forms as IXOR</td>
</tr>
<tr>
<td>ZABS</td>
<td>see ABS</td>
</tr>
<tr>
<td>ZCOS</td>
<td>see COS</td>
</tr>
<tr>
<td>ZEXP</td>
<td>see EXP</td>
</tr>
<tr>
<td>ZLOG</td>
<td>see “LOG(X)”</td>
</tr>
<tr>
<td>ZSIN</td>
<td>see SIN</td>
</tr>
<tr>
<td>ZSQRT</td>
<td>see SQRT</td>
</tr>
<tr>
<td>ZTAN</td>
<td>see TAN</td>
</tr>
</tbody>
</table>
Intrinsic Procedure Specifications

This section contains detailed specifications of Intel Fortran intrinsic procedures, which are listed in alphabetical order.

For a summary of all generic as well as specific intrinsic procedures, see Table 1-3 in the section “Generic and Specific Intrinsic Summary”.

All of the intrinsic procedures in this section are generic. This means that each intrinsic procedure may be called with more than one argument type/kind/rank pattern.

In many cases, the kind and type of intrinsic function results are the same as that of the “principal” argument. For example, the \texttt{SIN} function may be called with any kind of real argument or any kind of complex argument, and the result has the type and kind of the argument.

Intrinsic procedure references may use keywords, in which case the actual argument expression is preceded by the dummy argument name (the argument keyword) and an “=” symbol. These argument keywords are shown in the following descriptions of the procedures.

Some intrinsic procedure’s arguments are optional. Optional arguments are noted as such in the following descriptions.

\textbf{ABS}(A)

\begin{description}
\item[Description] Absolute value.
\item[Class] Elemental function.
\item[Argument] \(A\) must be of type integer, real, or complex.
\end{description}
Result Type and Type Parameter
The same as A except that if A is complex, the result is real.

Result Value
If A is of type integer or real, the value of the result is |A|.
If A is complex with value (x, y), the result is equal to a processor-dependent approximation to \( \sqrt{x^2 + y^2} \)

Examples
ABS(-1) has the value 1.
ABS(-1.5) has the value 1.5.
ABS((3.0, 4.0)) has the value 5.0.

ACHAR(I)
Converts an integer to an ASCII representation

Description
Returns the character in a specified position of the ASCII collating sequence. It is the inverse of the IACHAR function.

Class
Elemental function.

Argument
I must be of type integer.

Result Type and Type Parameter
Character of length one with kind type parameter value KIND(‘A’).
Result Value
If \( I \) has a value in the range \( 0 \leq I \leq 127 \), the result is the character in position \( I \) of the ASCII collating sequence, provided the processor is capable of representing that character; otherwise, the result is processor-dependent.

If the processor is not capable of representing both uppercase and lowercase letters and \( I \) corresponds to a letter in a case that the processor is not capable of representing, the result is the letter in the case that the processor is capable of representing.

\texttt{ACHAR(IACHAR(C))} must have the value \( C \) for any character \( C \) capable of representation in the processor.

Examples
\texttt{ACHAR(88)} is ‘X’.
\texttt{ACHAR(42)} is ‘*’.

\textbf{ACOS(X)}

Description
Arccosine (inverse cosine) function in radians.

Class
Elemental function.

Argument
\( x \) must be of type real with a value that satisfies the inequality \( |x| \leq 1 \).

Result Type and Type Parameter
Same as \( x \).
Result Value
The result has a value equal to a processor-dependent approximation to \( \arccos(X) \), expressed in radians. It lies in the range \( 0 \leq \text{ACOS}(X) \leq \pi \).

Examples
ACOS(0.54030231) has the value 1.0.
ACOS(.1_HIGH) has the value 1.4706289056333 with kind HIGH.

ACOSD(X)

Description
Arccosine (inverse cosine) in degrees.

Class
Elemental nonstandard function.

Argument
X must be of type real with a value that satisfies the inequality \( |X| \leq 1 \).

Result Type and Type Parameter
Same as X.

Result Value
The result has a value equal to a processor-dependent approximation to \( \arccos(X) \), expressed in degrees. It lies in the range \( 0 \leq \text{ACOSD}(X) \leq 180 \).

Examples
ACOSD(0.0000001) has the value 89.99999.
ACOSD(0.5) has the value 60.0.
ACOSD(-1.0) has the value 180.0.
ACOSH(X)

Description
Hyperbolic arccosine of radians.

Class
Elemental nonstandard function.

Argument
X must be of type real with a value $X \geq 1$.

Result Type and Type Parameter
Same as X.

Result Value
The result has a value equal to a processor-dependent approximation to the hyperbolic arccosine of X. It lies in the range $0 \leq \text{ACOSH}(X)$.

Examples
ACOSH(1.0) has the value 0.0.
ACOSH(180.0) has the value 5.8861.
ACOSH(0.0) has the value NaN (not a number).
ADJUSTL(STRING)

Description
Adjust to the left, removing leading blanks and inserting trailing blanks.

Class
Elemental function.

Argument
STRING must be of type character.

Result Type
Character of the same length and kind type parameter as STRING.

Result Value
The value of the result is the same as STRING except that any leading blanks have been deleted and the same number of trailing blanks have been inserted.

Example
ADJUSTL('bbWORD') is 'WORDbb'.
ADJUSTR(STRING)

Description
Adjust to the right, removing trailing blanks and inserting leading blanks.

Class
Elemental function.

Argument
STRING must be of type character.

Result Type
Character of the same length and kind type parameter as STRING.

Result Value
The value of the result is the same as STRING except that any trailing blanks have been deleted and the same number of leading blanks have been inserted.

Examples
ADJUSTR('WORDbb') has the value 'bbWORD'.
AIMAG(Z)

Description
Imaginary part of a complex number.

Class
Elemental function.

Argument
Z must be of type complex.

Result Type and Type Parameter
Real with the same kind type parameter as Z.

Result Value
If Z has the value (x, y), the result has value y.

Examples
AIMAG((2.0, 3.0)) has the value 3.0.
AIMAG((2.0_HIGH, 3.0)) has the value 3.0 with kind HIGH; the parts of a complex literal constant have the same precision, which is that of the part with the greater precision.
AINT(A, KIND)

Optional Argument
KIND

Description
Truncation to a whole number.

Class
Elemental function.

Arguments
A must be of type real.
KIND (optional) must be a scalar integer initialization expression.

Result Type and Type Parameter
The result is of type real. If KIND is present, the kind type parameter is that specified by KIND; otherwise, the kind type parameter is that of A.

Result Value
If |A| < 1, AINT(A) has the value 0; if A ≥ 1, AINT(A) has a value equal to the integer whose magnitude is the largest integer that does not exceed the magnitude of A and whose sign is the same as the sign of A.

Examples
AINT(2.783) has the value 2.0.
AINT(-2.783) has the value -2.0.
AINT(2.11111111111111, HIGH) and AINT(2.11111111111111, HIGH) have the value 2.0 with kind HIGH.
ALL(MASK, DIM)

Optional Argument
DIM

Description
Determine whether all values are .TRUE. in MASK along dimension DIM.

Class
Transformational function.

Arguments
MASK
must be of type logical. It must not be scalar.

DIM (optional)
must be scalar and of type integer with value in the range $1 \leq DIM \leq n$
where $n$ is the rank of MASK. The corresponding actual argument must not
be an optional dummy argument.

Result Type, Type Parameter, and Shape
The result is of type logical with the same kind type parameter as MASK. It
is scalar if DIM is absent or MASK has rank one; otherwise, the result is an
array of rank $n-1$ and of shape $(d_1, d_2, \ldots, d_{DIM-1}, d_{DIM+1}, \ldots, d_n)$ where $(d_1,
\ldots, d_n)$ is the shape of MASK.
Result Value

Case 1  The result of \texttt{ALL(MASK)} has the value \texttt{.TRUE.} if all elements of \texttt{MASK} are \texttt{.TRUE.} or if \texttt{MASK} has size zero, and the result has value \texttt{.FALSE.} if any element of \texttt{MASK} is \texttt{.FALSE.}.

Case 2  If \texttt{MASK} has rank one, \texttt{ALL(MASK, DIM)} has a value equal to that of \texttt{ALL(MASK)}. Otherwise, the value of element \((s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_n)\) of \texttt{ALL(MASK, DIM)} is equal to \texttt{ALL(MASK \((s_1, s_2, \ldots, s_{DIM-1}, ::, s_{DIM+1}, \ldots, s_n)\)).

Examples

Case 1  The value of \texttt{ALL((/ .TRUE., .FALSE., .TRUE. /))} is \texttt{.FALSE.}.

\[
\text{ALL((/ .TRUE._BIT, .TRUE._BIT, .TRUE._BIT /)) is the value .TRUE._BIT.}
\]

Case 2  If \(B\) is the array

\[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix}
\]

and \(C\) is the array

\[
\begin{bmatrix}
0 & 3 & 5 \\
7 & 4 & 8
\end{bmatrix}
\]

then \texttt{ALL(B .NE. C, DIM = 1)} is \texttt{[.TRUE., .FALSE., .FALSE., .FALSE.]} and \texttt{ALL(B .NE. C, DIM = 2)} is \texttt{[.FALSE., .FALSE., .FALSE.].}
ALLOCATED(ARRAY)

Description
Indicate whether or not an allocatable array is currently allocated.

Class
Inquiry function.

Argument
ARRAY must be an allocatable array.

Result Type, Type Parameter, and Shape
Default logical scalar.

Result Value
The result has the value .TRUE. if ARRAY is currently allocated and has the value .FALSE. if ARRAY is not currently allocated. The result is undefined if the allocation status of the array is undefined.

Example
If the following statements are processed
REAL, ALLOCATABLE :: A(:, :)
ALLOCATE (A(10, 10))
PRINT *, ALLOCATED (A)
then T is printed.
AND(I, J)

Description
Bitwise AND.

Class
Elemental nonstandard function.

Arguments
I must be of type integer.
J must be of type integer with the same kind type parameter as I.

Result Type and Type Parameter
Same as I.

Result Value
The result has the value obtained by performing a bitwise AND on I and J according to the following truth table:

<table>
<thead>
<tr>
<th>I</th>
<th>J</th>
<th>AND(I, J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
The model for interpreting an integer value as a sequence of bits is in the section “The Bit Model”.

Example
The following program produces the output “1151”.
K is assigned the binary value 0001, which is 1 in decimal.

```fortran
PROGRAM andtest
   INTEGER I, J, K
   I = B’1011’
   J = B’0101’
   K = AND(I, J)
   PRINT *, I, J, K
END
```

**ANINT(A, KIND)**

**Optional Argument**

KIND

**Description**

Nearest whole number.

**Class**

Elemental function.

**Arguments**

A must be of type real.

KIND (optional) must be a scalar integer initialization expression.
Result Type and Type Parameter

The result is of type real. If KIND is present, the kind type parameter is that specified by KIND; otherwise, the kind type parameter is that of A.

Result Value

If A>0, ANINT(A) has the value AINT(A+0.5); if A≤0, ANINT(A) has the value AINT(A–0.5).

Examples

ANINT(2.783) has the value 3.0.
ANINT(–2.783) has the value –3.0.
ANINT(2.7837837837837_HIGH) and
ANINT(2.7837837837837,HIGH) have the value 3.0 with kind HIGH.

ANY(MASK, DIM)

Optional Argument

DIM

Description

Determine whether any value is .TRUE. in MASK along dimension DIM.

Class

Transformational function.

Arguments

MASK must be of type logical. It must not be scalar.
DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of MASK. The corresponding actual argument must not be an optional dummy argument.

**Result Type, Type Parameter, and Shape**

The result is of type logical with the same kind type parameter as MASK. It is scalar if DIM is absent or MASK has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of MASK.

**Result Value**

**Case 1**

The result of ANY(MASK) has the value .TRUE. if any element of MASK is .TRUE. and has the value .FALSE. if no elements are .TRUE. or if MASK has size zero.

**Case 2**

If MASK has rank one, ANY(MASK, DIM) has a value equal to that of ANY(MASK). Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of ANY(MASK, DIM) is equal to ANY(MASK(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)).

**Examples**

**Case 1**

The value of ANY((/ .TRUE., .FALSE., .TRUE. /)) is .TRUE..

ANY((/ .FALSE._BIT, .FALSE._BIT, .FALSE._BIT /)) is .FALSE._BIT.

**Case 2**

If B is the array

\[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix}
\]

and C is the array

\[
\begin{bmatrix}
0 & 3 & 5 \\
7 & 4 & 8
\end{bmatrix}
\]
then \( \text{ANY}(B \ .\text{NE.} \ C, \ \text{DIM} = 1) \) is \([.\text{TRUE.}, \ .\text{FALSE.}, .\text{TRUE.}]\) and \( \text{ANY}(B \ .\text{NE.} \ C, \ \text{DIM} = 2) \) is \([.\text{TRUE.}, .\text{TRUE.}]\).

---

**ASIN(X)**

**Description**

Arcsine (inverse sine) function in radians.

**Class**

Elemental function.

**Argument**

\( x \) must be of type real. Its value must satisfy the inequality \(|x| \leq 1\).

**Result Type and Type Parameter**

Same as \( x \).

**Result Value**

The result has a value equal to a processor-dependent approximation to \( \text{arcsin}(x) \), expressed in radians. It lies in the range \(-\pi/2 \leq \text{ASIN}(x) \leq \pi/2\).

**Examples**

\( \text{ASIN}(0.84147098) \) has the value \( 1.0 \).
\( \text{ASIN}(1.0\_\text{HIGH}) \) has the value \( 1.5707963267949 \) with kind \( \text{HIGH} \).
ASIND(X)

Description
Arcsine (inverse sine) function in degrees.

Class
Elemental nonstandard function.

Argument
X must be of type real. Its value must satisfy the inequality |x| ≤ 1.

Result Type and Type Parameter
Same as x.

Result Value
The result has a value equal to a processor-dependent approximation to arcsin(x), expressed in degrees. It lies in the range -90 ≤ ASIND(X) ≤ 90.

Examples
ASIND(-1.0) has the value -90.0.
ASIND(0.5) has the value 30.0.
**ASINH(X)**

**Description**
Hyperbolic arcsine of radians.

**Class**
Elemental nonstandard function.

**Argument**
x must be of type real.

**Result Type and Type Parameter**
Same as x.

**Result Value**
The result has a value equal to a processor-dependent approximation to the hyperbolic arcsine of x.

**Examples**
- \( \text{ASINH}(1.0) \) has the value \(-0.88137\).
- \( \text{ASINH}(180.0) \) has the value \(5.88611\).
ASSOCIATED(POINTER, TARGET)

Optional Argument
TARGET

Description
Returns the association status of its pointer argument or indicates the pointer is associated with the target.

Class
Inquiry function.

Arguments
POINTER must be a pointer and may be of any type. Its pointer association status must not be undefined.
TARGET (optional) must be a pointer or target. If it is a pointer, its pointer association status must not be undefined.

Result Type
The result is scalar of type default logical.

Result Value
Case 1 If TARGET is absent, the result is .TRUE. if POINTER is currently associated with a target and .FALSE. if it is not.
Case 2 If TARGET is present and is a target, the result is .TRUE. if POINTER is currently associated with TARGET and .FALSE. if it is not.
Case 3 If TARGET is present and is a pointer, the result is .TRUE. if both POINTER and TARGET are currently associated with the same target, and is .FALSE. otherwise. If either POINTER or TARGET is disassociated, the result is .FALSE..

Examples
Case 1 ASSOCIATED(PTR) is .TRUE. if PTR is currently associated with a target.

Case 2 ASSOCIATED(PTR, TAR) is .TRUE. if the following statements have been processed:

```
REAL, TARGET :: TAR (0:100)
REAL, POINTER :: PTR(:)
PTR => TAR
```

The subscript range for both TAR and PTR is 0:100.

If the pointer assignment statement is either

```
PTR => TAR(:)
```
or

```
PTR => TAR(0:100)
```

then ASSOCIATED(PTR, TAR) is still .TRUE., but in both cases the subscript range for PTR is 1:101.

However, if the pointer assignment statement is

```
PTR => TAR(0:99)
```

then ASSOCIATED(PTR, TAR) is .FALSE., because TAR(0:99) is not the same as TAR.
Case 3  \[ \text{ASSOCIATED}(	ext{PTR1}, \text{PTR2}) \text{ is } \text{.TRUE. if the} \]
\[ \text{following statements have been processed.} \]
REAL, POINTER :: PTR1(:), PTR2(:)
ALLOCATE(PTR1(0:10))
PTR2 => PTR1
After the execution of either
NULLIFY(PTR1)
or
NULLIFY(PTR2)
the statement \text{ASSOCIATED}(	ext{PTR1}, \text{PTR2}) \text{ evaluates to} \]
\[ \text{.FALSE..} \]

\textbf{ATAN(X)}

\textbf{Description}
Arctangent (inverse tangent) function in radians.

\textbf{Class}
Elemental function.

\textbf{Argument}
x must be of type real.

\textbf{Result Type and Type Parameter}
Same as x.
Result Value
The result has a value equal to a processor-dependent approximation to arctan(X), expressed in radians, that lies in the range $-\pi/2 \leq \text{ATAN}(X) \leq \pi/2$.

Examples
\text{ATAN}(1.5574077)\) has the value 1.0.
\text{ATAN}(2.0\_\text{HIGH}/3.0) has the value 0.58800260354757 with kind \text{HIGH}.

\textbf{ATAN2}(Y, X)

Description
Arctangent (inverse tangent) function in radians. The result is the principal value of the argument of the nonzero complex number (X, Y).

Class
Elemental function.

Arguments
\begin{align*}
Y & \quad \text{must be of type real.} \\
X & \quad \text{must be of the same type and kind type parameter as } Y. \\
& \quad \text{If } y \text{ has the value zero, } x \text{ must not have the value zero.}
\end{align*}

Result Type and Type Parameter
Same as \(x\).
Result Value

The result has a value equal to a processor-dependent approximation to the principal value of the argument of the complex number \((X, Y)\), expressed in radians.

The result lies in the range \(-\pi \leq \text{ATAN2}(Y, X) \leq \pi\) and is equal to a processor-dependent approximation to a value of \(\text{arctan}(Y/X)\) if \(X\neq 0\).

If \(Y>0\), the result is positive. If \(Y=0\), the result is zero if \(X>0\) and the result is \(\pi\) if \(X<0\). If \(Y<0\), the result is negative. If \(X=0\), the absolute value of the result is \(\pi/2\).

Examples

\(\text{ATAN2}(1.5574077, 1.0)\) has the value 1.0.

If \(Y\) has the value
\[
\begin{bmatrix}
1 & 1 \\
\text{D}1 & \text{D}1
\end{bmatrix}
\]
and \(X\) has the value
\[
\begin{bmatrix}
\text{D}1 & 1 \\
\text{D}1 & 1
\end{bmatrix}
\]
then the value of \(\text{ATAN2}(Y, X)\) is
\[
\begin{bmatrix}
\frac{3\pi}{4} & \frac{\pi}{4} \\
\frac{3\pi}{4} & \frac{\pi}{4}
\end{bmatrix}
\]
ATAN2D(Y, X)

Description
Arctangent (inverse tangent) function in degrees.

Class
Elemental nonstandard function.

Arguments
Y must be of type real.
X must be of the same type and kind type parameter as Y.

Result Type and Type Parameter
Same as X.

Result Value
The result has a value equal to a processor-dependent approximation to the principal value of the argument of the complex number (X, Y), expressed in degrees, that lies in the range -90 < ATAN2D(Y, X) < 90.

Examples
ATAN2D(1.0, 1.0) has the value 45.0.
ATAN2D(1.0, 0.0) has the value 90.0.
ATAN2D(8735.0, 1.0) has the value 89.99344.
ATAND(X)

Description
Arctangent (inverse tangent) function in degrees.

Class
Elemental nonstandard function.

Argument
X must be of type real.

Result Type and Type Parameter
Same as X.

Result Value
The result has a value equal to a processor-dependent approximation to arctan(X), expressed in degrees, that lies in the range \(-90 < ATAND(X) < 90\).

Examples
ATAND(1.0) has the value 45.0.
ATAND(0.0) has the value 0.0.
ATAND(-94373.0) has the value -89.9994.
**ATANH(X)**

**Description**
Hyperbolic arctangent of radians.

**Class**
Elemental nonstandard function.

**Argument**
X must be of type real.

**Result Type and Type Parameter**
Same as x.

**Result Value**
The result has a value equal to a processor-dependent approximation to the hyperbolic arctangent of X.

**Examples**
- ATANH (0.0) has the value 0.0.
- ATANH (-0.77) has the value -1.02033.
- ATANH (0.5) has the value 0.549306.
BADDRESS(X)

Description
Return the address of X.

Class
Inquiry nonstandard function.

Argument
X may be of any type.

Result Type
The result is of type default integer.

Example.
The following program:
PROGRAM batest
   INTEGER X(5), I
   DO I=1, 5
      PRINT *, BADDRESS(X(I))
   END DO
END
Could produce this output:
2063835808
2063835812
2063835816
2063835820
2063835824
BIT_SIZE(I)

**Description**
Returns the number of bits $n$, defined by the model in the section “The Bit Model”, for integers with the kind parameter of the argument.

**Class**
Inquiry function.

**Argument**
$I$ must be of type integer.

**Result Type, Type Parameter, and Shape**
Scalar integer with the same kind type parameter as $I$.

**Result Value**
The result has the value of the number of bits $n$ in the model integer, defined for bit manipulation contexts in the section “The Bit Model”, for integers with the kind parameter of the argument.

**Examples**
BIT_SIZE(1) has the value 32 if $n$ in the model is 32.
BTEST(I, POS)

**Description**
Tests a bit of an integer value.

**Class**
Elemental function.

**Argument**
- **I** must be of type integer.
- **POS** must be of type integer. It must be nonnegative and be less than BIT_SIZE(I).

**Result Type**
The result is of type default logical.

**Result Value**
The result has the value .TRUE. if bit POS of I has the value 1 and has the value .FALSE. if bit POS of I has the value 0. The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

**Examples**
- BTEST(8, 3) has the value .TRUE..
- BTEST(8_SHORT, 3) has the value .TRUE..

If A has the value [1, 2, 3, 4] then the value of BTEST(A, 2) is [.FALSE., .FALSE., .FALSE., .TRUE.] and the value of BTEST(2, A) is [.TRUE., .FALSE., .FALSE., .FALSE., .FALSE.].
CEILING(A)

Description
Returns the least integer greater than or equal to its argument.

Class
Elemental generic function.

\[
\text{result} = \text{CEILING} \ (A, \ \text{KIND})
\]

Arguments
- \(A\) Input. Must be of type real.
- \(\text{KIND} \) (optional) Input. Must be a scalar integer initialization expression; a Fortran 95 feature.

Result Type and Type Parameter
Default integer.

Result Value
If \(\text{KIND}\) is present, the kind parameter is that specified by \(\text{KIND}\). Otherwise, the kind parameter is that of default integer. The result has a value equal to the least integer greater than or equal to \(A\). The result is undefined if the processor cannot represent this value in the default integer type.

Examples
- \(\text{CEILING}(3.7)\) has the value 4.
- \(\text{CEILING}(-3.7)\) has the value –3.
- \(\text{CEILING}(20.0_{\text{HIGH}}/3)\) has the value 7.
CHAR(I, KIND)

Optional Argument
KIND

Description
Returns the character in a given position of the processor collating sequence associated with the specified kind type parameter. It is the inverse of the function ICHAR.

Class
Elemental function.

Arguments
I must be of type integer with a value in the range \(0 \leq I \leq n-1\), where \(n\) is the number of characters in the collating sequence associated with the specified kind type parameter.

KIND (optional) must be a scalar integer initialization expression.

Result Type and Type Parameters
Character of length one. If KIND is present, the kind type parameter is that specified by KIND; otherwise, the kind type parameter is that of default character type.
Result Value

The result is the character in position \( i \) of the collating sequence associated with the specified kind type parameter.

\[
\text{ICHAR(CHAR(I, KIND(C))))}
\]

must have the value \( I \) for \( 0 \leq I \leq n-1 \) and

\[
\text{CHAR(ICHAR(C), KIND(C))}
\]

must have the value \( C \) for any character \( C \) capable of representation in the processor.

Example.

\[
\text{CHAR(88)}
\]

is 'X' on a processor using the ASCII collating sequence.

\[\text{CMPLX(X, Y, KIND)}\]

Optional Arguments

\( Y, \text{KIND} \)

Description

Convert to complex type.

Class

Elemental function.

Arguments

\( X \) must be of type integer, real, or complex.

\( Y \) (optional) must be of type integer or real. It must not be present if \( X \) is of type complex.

\( \text{KIND} \) (optional) must be a scalar integer initialization expression.
Result Type and Type Parameter
The result is of type complex. If KIND is present, the kind type parameter is that specified by KIND; otherwise, the kind type parameter is that of default real type.

Result Value
If Y is absent and X is not complex, it is as if Y were present with the value zero.
If Y is absent and X is complex, it is as if Y were present with the value AIMAG(X).
CMPLX(X,Y,KIND) has the complex value whose real part is REAL(X,KIND) and whose imaginary part is REAL(Y,KIND).

Examples
CMPLX(-3) is -3.0 + 0i.
CMPLX((4.1,0.0), KIND=HIGH), CMPLX((4.1,0), KIND=HIGH), and CMPLX(4.1, KIND=HIGH) are each 4.1 + 0.0i with kind HIGH.

CONJG(Z)

Description
Conjugate of a complex number.

Class
Elemental function.

Argument
Z must be of type complex.
Result Type and Type Parameter
Same as z.

Result Value
If z has the value (x, y), the result has the value (x, -y).

Examples
CONJG((2.0, 3.0)) is (2.0, -3.0).
CONJG((0., -4.1_HIGH)) is 0 + 4.1i with kind HIGH.

COS(X)

Description
Cosine function in radians.

Class
Elemental function.

Argument
x must be of type real or complex.

Result Type and Type Parameter
Same as x.
Result Value
The result has a value equal to a processor-dependent approximation to cos(x). If x is of type real, it is regarded as a value in radians. If x is of type complex, its real part is regarded as a value in radians.

Examples
\( \cos(1.0) \) has the value 0.54030231.
\( \cos((1.0_{\text{HIGH}}, 1.0)) \) has the value:
0.83373002513115 – 0.98889770576287i with kind \text{HIGH}.

\textbf{COSD(X)}

\underline{Description}
Cosine function that accepts input in degrees.

\underline{Class}
Elemental nonstandard function.

\underline{Argument}
x must be of type real.

\underline{Result Type and Type Parameter}
Same as x.
Result Value
The result has a value equal to a processor-dependent approximation to \( \cos(X) \).

Examples
\[ \cosd(0.0) \] has the value 1.0.
\[ \cosd(60.0) \] has the value 0.5.

**COSH(X)**

Description
Hyperbolic cosine function.

Class
Elemental function.

Argument
\( X \) must be of type real.

Result Type and Type Parameter
Same as \( X \).

Result Value
The result has a value equal to a processor-dependent approximation to \( \cosh(X) \).

Examples
\[ \cosh(1.0) \] has the value 1.5430806.
\[ \cosh(0.1_{\text{HIGH}}) \] has the value 1.0050041680558 with kind \text{HIGH}. 

COUNT(MASK, DIM)

Optional Argument

DIM

Description

Count the number of .TRUE. elements of MASK along dimension DIM.

Class. Transformational function.

Argument

MASK must be of type logical. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range

\[1 \leq \text{DIM} \leq n\]

where \(n\) is the rank of \(\text{MASK}\). The corresponding actual argument must not be an optional dummy argument.

Result Type, Type Parameter, and Shape

The result is of type default integer. It is scalar if \(\text{DIM}\) is absent or \(\text{MASK}\) has rank one; otherwise, the result is an array of rank \(n-1\) and of shape \((d_1, d_2, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)\) where \((d_1, d_2, \ldots, d_n)\) is the shape of \(\text{MASK}\).
Result Value

Case 1  The result of COUNT(MASK) has a value equal to the number of .TRUE. elements of MASK or has the value zero if MASK has size zero.

Case 2  If MASK has rank one, COUNT(MASK, DIM) has a value equal to that of COUNT(MASK). Otherwise, the value of element \((s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)\) of COUNT(MASK, DIM) is equal to COUNT(MASK \((s_1, s_2, \ldots, s_{\text{DIM}-1}, : , s_{\text{DIM}+1}, \ldots, s_n)\)).

Examples

Case 1  The value of COUNT((/ .TRUE., .FALSE., .TRUE. /)) is 2.

Case 2  If B is the array

\[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6 \\
\end{bmatrix}
\]

and C is the array

\[
\begin{bmatrix}
0 & 3 & 5 \\
7 & 4 & 6 \\
\end{bmatrix}
\]

then COUNT(B .NE. C, DIM = 1) is [2, 0, 1] and COUNT(B .NE. C, DIM = 2) is [1, 2].
CPU_TIME(TIME)

Description
CPU_TIME returns a processor dependent time in seconds. To get elapsed CPU_TIME, you must call the intrinsic twice, once to get the start time, and again to get a finish time, and then subtract start from finish.

Class
Subroutine

Argument
TIME must be scalar and of type real.
As an extension Intel Fortran allows TIME to be of type REAL*8, REAL*16, or DOUBLE-PRECISION.

Result Type and Type Parameter
Same as TIME.

Example
PROGRAM TIMEIT
  REAL STARTTIME/0.0/,STOPTIME/0.0/
  A = 1.2
  CALL CPU_TIME(STARTTIME)
  DO I = 1,100000
    B = CCOS(CMPLX(a,0.0)) * CSIN(CMPLX(a,0.0))
  ENDDO
  CALL CPU_TIME(STOPTIME)
  PRINT *,'CPU TIME WAS ',STOPTIME - STARTTIME
END
CSHIFT(ARRAY, SHIFT, DIM)

Optional Argument

DIM

Description

Perform a circular shift on an array expression of rank one, or perform circular shifts on all the complete rank one sections along a given dimension of an array expression of rank two or greater.

Elements shifted out at one end of a section are shifted in at the other end. Different sections may be shifted by different amounts and in different directions (positive for left shifts, negative for right shifts).

Class

Transformational function.

Arguments

ARRAY may be of any type. It must not be scalar.

SHIFT must be of type integer and must be scalar if ARRAY has rank one; otherwise, it must be scalar or of rank n-1 and of shape \((d_1, d_2, \ldots, d_{DIM-1}, d_{DIM+1}, \ldots, d_n)\) where \((d_1, d_2, \ldots, d_n)\) is the shape of ARRAY.

DIM (optional) must be a scalar and of type integer with a value in the range \(1 \leq DIM \leq n\), where \(n\) is the rank of ARRAY. If DIM is omitted, it is as if it were present with the value 1.

Result Type, Type Parameter, and Shape

The result is of the type and type parameters of ARRAY, and has the shape of ARRAY.
Result Value

Case 1  If ARRAY has rank one, element i of the result is

\[
\text{ARRAY}(1 + \text{MODULO}(i + \text{SHIFT} - 1, \text{SIZE}(\text{ARRAY})))
\]

Case 2  If ARRAY has rank greater than one, section \((s_1, s_2, \ldots, s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, \ldots, s_n)\) of the result has a value equal to \(\text{CSHIFT}(\text{ARRAY}(s_1, s_2, \ldots, s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, \ldots, s_n), \text{sh}, 1)\), where \(\text{sh}\) is \(\text{SHIFT}\) or \(\text{SHIFT}(s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)\).

Examples

Case 1  If \(V\) is the array \([1, 2, 3, 4, 5, 6]\), the effect of shifting \(V\) circularly to the left by two positions is achieved by \(\text{CSHIFT}(V, \text{SHIFT} = 2)\) which has the value \([3, 4, 5, 6, 1, 2]\).

\(\text{CSHIFT}(V, \text{SHIFT} = -2)\) achieves a circular shift to the right by two positions and has the value \([5, 6, 1, 2, 3, 4]\).

Case 2  The rows of an array of rank two may all be shifted by the same amount or by different amounts. If \(M\) is the array

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
\]

then the value of \(\text{CSHIFT}(M, \text{SHIFT} = -1, \text{DIM} = 2)\) is

\[
\begin{pmatrix}
3 & 1 & 2 \\
6 & 4 & 5 \\
9 & 7 & 8
\end{pmatrix}
\]

and the value of \(\text{CSHIFT}(M, \text{SHIFT} = (/ -1, 1, 0 /), \text{DIM} = 2)\) is
DATE_AND_TIME(DATE, TIME, ZONE, VALUES)

Optional Arguments

DATE, TIME, ZONE, VALUES

Description

Returns data on the real-time clock and date in a form compatible with the representations defined in ISO 8601:1988 (“Data elements and interchange formats — Information interchange — Representation of dates and times”).

Class

Subroutine.

Arguments

DATE (optional) must be scalar and of type default character, and must be of length at least 8 in order to contain the complete value. It is an INTENT (OUT) argument. Its leftmost 8 characters are set to a value of the form CCYYMMDD, where CC is the century, YY the year within the century, MM the month within the year, and DD the day within the month. If there is no date available, they are set to blank.

TIME (optional) must be scalar and of type default character, and must be of length at least 10 in order to contain the complete value. It is an INTENT (OUT) argument. Its leftmost 10 characters are set to a value of the form hhmmss.sss, where hh is the hour of the day, mm is the minutes of the...
hour, and \texttt{ss.sss} is the seconds and milliseconds of the minute. If there is no clock available, they are set to blank.

\texttt{ZONE} (optional) must be scalar and of type default character, and must be of length at least 5 in order to contain the complete value. It is an \texttt{INTENT(OUT)} argument. Its leftmost 5 characters are set to a value of the form $\pm hhmm$, where $hh$ and $mm$ are the time difference with respect to Coordinated Universal Time (UTC) in hours and parts of an hour expressed in minutes, respectively. If there is no clock available, they are set to blank.

\texttt{VALUES} (optional) must be of type default integer and of rank one. It is an \texttt{INTENT(OUT)} argument. Its size must be at least 8. The values returned in \texttt{VALUES} are as follows:

\texttt{VALUES(1)} the year (for example, 1990), or \texttt{HUGE(0)} if there is no date available;

\texttt{VALUES(2)} the month of the year, or \texttt{HUGE(0)} if there is no date available;

\texttt{VALUES(3)} the day of the month, or \texttt{HUGE(0)} if there is no date available;

\texttt{VALUES(4)} the time difference with respect to Coordinated Universal Time (UTC) in minutes, or \texttt{HUGE(0)} if this information is not available;

\texttt{VALUES(5)} the hour of the day, in the range of 0 to 23, or \texttt{HUGE(0)} if there is no clock;

\texttt{VALUES(6)} the minutes of the hour, in the range 0 to 59, or \texttt{HUGE(0)} if there is no clock;

\texttt{VALUES(7)} the seconds of the minute, in the range 0 to 60, or \texttt{HUGE(0)} if there is no clock;

\texttt{VALUES(8)} the milliseconds of the second, in the range 0 to 999, or \texttt{HUGE(0)} if there is no clock.

The \texttt{HUGE} intrinsic function is described in the section “\texttt{HUGE(X)}”.
Example

INTEGER DATE_TIME (8)
CHARACTER (LEN = 10) BIG_BEN (3)
CALL DATE_AND_TIME (BIG_BEN (1), BIG_BEN (2), &
BIG_BEN (3), DATE_TIME)

if called in Geneva, Switzerland on 1985 April 12 at 15:27:35.5 would have
assigned the value "19850412bb" to BIG_BEN(1), the value "152735.500"
to BIG_BEN(2), and the value "+0100bbbb" to BIG_BEN(3), and the
following values to DATE_TIME: 1985, 4, 12, 60, 15, 27, 35, 500.

Note that UTC is defined by CCIR Recommendation 460-2 (and is also
known as Greenwich Mean Time).

**DBLE(A)**

**Description**
Convert to double precision real type.

**Class**
Elemental function.

**Argument**
A must be of type integer, real, or complex.

**Result Type and Type Parameter**
Double precision real.

**Result Value**
Case 1 If A is of type double precision real, DBLE (A) = A.
Case 2  If $A$ is of type integer or real, the result is as much precision of the significant part of $A$ as a double precision real datum can contain.

Case 3  If $A$ is of type complex, the result is as much precision of the significant part of the real part of $A$ as a double precision real datum can contain.

**Examples**

- \texttt{DBLE(-0.3)} is $-0.3$ of type double precision real.
- \texttt{DBLE(1.0\_HIGH/3)} is $0.33333333333333$ of type double precision real.

---

**DFLOAT(A)**

**Description**

Convert to double precision type.

**Class**

Elemental nonstandard function.

**Argument**

$A$ must be of type integer.

**Result Type and Type Parameter**

Double precision.

**Example**

\texttt{DFLOAT(56)} is $56.0$ of type double precision.
**DIGITS(X)**

**Description**
Returns the number of significant digits in the model representing numbers of the same type and kind type parameter as the argument.

**Class**
Inquiry function.

**Argument**
X must be of type integer or real. It may be scalar or array valued.

**Result Type, Type Parameter, and Shape**
Default integer scalar.

**Result Value**
The result has the value q if X is of type integer and p if X is of type real, where q and p are as defined in the section “Data Representation Models” for the model representing numbers of the same type and kind type parameter as X.

**Example**
DIGITS(X) has the value 24 for real X whose model is described in the section “The Real Number System Model”.

**DIM(X, Y)**

**Description**
The difference $X - Y$ if it is positive; otherwise zero.

**Class**
Elemental function.

**Argument**
- $X$ must be of type integer or real.
- $Y$ must be of the same type and kind type parameter as $X$.

**Result Type and Type Parameter**
Same as $X$.

**Result Value**
The value of the result is $X - Y$ if $X > Y$ and zero otherwise.

**Examples**
- $\text{DIM}(5, 3)$ has the value 2. $\text{DIM}(-3.0, 2.0)$ has the value 0.0.
DNUM(I)

**Description**
Convert to double precision.

**Class**
Elemental nonstandard function.

**Argument**
I must be of type character.

**Result Type**
Double precision.

**Examples**
DNUM("3.14159") is 3.14159 of type double precision.
The following code sets x to 311.0:
```fortran
  CHARACTER(3) i
  DOUBLE PRECISION x
  i = "311"
  x = DNUM(i)
```
DOT_PRODUCT(VECTOR_A, VECTOR_B)

**Description**
Performs dot-product multiplication of numeric or logical vectors.

**Class**
Transformational function.

**Argument**
- **VECTOR_A** must be of numeric type (integer, real, or complex) or of logical type. It must be array valued and of rank one.
- **VECTOR_B** must be of numeric type if VECTOR_A is of numeric type or of type logical if VECTOR_A is of type logical. It must be array valued and of rank one. It must be of the same size as VECTOR_A.

**Result Type, Type Parameter, and Shape**
The result is scalar.
If the arguments are of numeric type, the type and kind type parameter of the result are those of the expression VECTOR_A * VECTOR_B determined by the types of the arguments.
If the arguments are of type logical, the result is of type logical with the kind type parameter of the expression VECTOR_A .AND. VECTOR_B.
Intrinsic Procedures

Result Value

Case 1  If \texttt{VECTOR\_A} is of type integer or real, the result has the value \(\text{SUM(VECTOR\_A*VECTOR\_B)}\). If the vectors have size zero, the result has the value zero.

Case 2  If \texttt{VECTOR\_A} is of type complex, the result has the value \(\text{SUM(CONJG(VECTOR\_A)*VECTOR\_B)}\). If the vectors have size zero, the result has the value zero.

Case 3  If \texttt{VECTOR\_A} is of type logical, the result has the value \(\text{ANY(VECTOR\_A .AND. VECTOR\_B)}\). If the vectors have size zero, the result has the value \(\text{.FALSE.}\).

Examples

Case 1  \texttt{DOT\_PRODUCT((/ 1, 2, 3 /), (/ 2, 3, 4 /))} has the value 20.

Case 2  \texttt{DOT\_PRODUCT((/ (1.0, 2.0), (2.0, 3.0) /), (/ (1.0, 1.0), (1.0, 4.0) /))} has the value 17 + 4i.

Case 3  \texttt{DOT\_PRODUCT((/ .TRUE., .FALSE. /), (/ .TRUE., .TRUE. /))} has the value \(\text{.TRUE.}\).

DPROD(X, Y)

Description

Double precision real product.

Class

Elemental function.
Argument

\( X \) must be of type default real.
\( Y \) must be of type default real.

Result Type and Type Parameters
Double precision real.

Result Value
The result has a value equal to a processor-dependent approximation to the product of \( X \) and \( Y \).

Example
\[ \text{DPROD} (-3.0, 2.0) \] has the value \(-6.0\) of type double precision real.

---

DREAL(A)

Description
Convert to double precision.

Class
Elemental nonstandard function.

Argument
\( A \) must be of type integer, real, or complex.

Result
Double precision.
Examples

DREAL(91) is 91.0 of type double precision.

The following code sets x to 45.34:

```fortran
COMPLEX p
DOUBLE PRECISION x
p = (45.34, 1.0)
x = DREAL(p)
```

DSIGN

*Returns the absolute value multiplied by the sign of the second argument*

Description

This function performs a sign transfer by returning the absolute value of the first argument multiplied by the sign of the second argument.

Class

Non-standard elemental function.

Prototype

Arguments

A1 number whose absolute value is the magnitude of the result

A2 number whose sign is the sign of the result

Result

| A1 | if A2>=0  
-|A1 | if b<0

If A2 is zero,
if the process cannot distinguish between $+0$ and $-0$,
the result is $|A_1|$
otherwise, for $-0$, the value is $-|A_1|$
for $+0$, the value is $|A_1|$

**EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)**

**Optional Argument**

BOUNDARY, DIM

**Description**

Perform an end-off shift on an array expression of rank one or perform end-off shifts on all the complete rank-one sections along a given dimension of an array expression of rank two or greater.

Elements are shifted off at one end of a section and copies of a boundary value are shifted in at the other end.

Different sections may have different boundary values and may be shifted by different amounts and in different directions (positive for left shifts, negative for right shifts).

**Class**

Transformational function.

**Argument**

*ARRAY* may be of any type. It must not be scalar.

*SHIFT* must be of type integer and must be scalar if ARRAY has rank one; otherwise, it must be scalar or of rank $n-1$ and of shape $(d_1, d_2, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)$ where $(d_1, d_2, \ldots, d_n)$ is the shape of ARRAY.
BOUNDARY (optional) must be of the same type and type parameters as ARRAY and must be scalar if ARRAY has rank one; otherwise, it must be either scalar or of rank \( n-1 \) and of shape \((d_1, d_2, \ldots, d_{DIM-1}, d_{DIM+1}, \ldots, d_n)\). BOUNDARY may be omitted for the data types in the following table and, in this case, it is as if it were present with the scalar value shown.

<table>
<thead>
<tr>
<th>Type of ARRAY</th>
<th>Value of BOUNDARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>0</td>
</tr>
<tr>
<td>Real</td>
<td>0.0</td>
</tr>
<tr>
<td>Complex</td>
<td>(0.0, 0.0)</td>
</tr>
<tr>
<td>Logical</td>
<td>.FALSE.</td>
</tr>
<tr>
<td>Character (len)</td>
<td>( len ) blanks</td>
</tr>
</tbody>
</table>

DIM (optional) must be scalar and of type integer with a value in the range \( 1 \leq \text{DIM} \leq n \), where \( n \) is the rank of ARRAY. If DIM is omitted, it is as if it were present with the value 1.

**Result Type, Type Parameter, and Shape**

The result has the type, type parameters, and shape of ARRAY.

**Result Value**

Element \((s_1, s_1, \ldots, s_n)\) of the result has the value ARRAY \((s_1, s_2, \ldots, s_{DIM-1}, s_{DIM} + sh, s_{DIM+1}, \ldots, s_n)\) where \( sh \) is SHIFT or SHIFT\((s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_n)\) provided the inequality \( \text{LBOUND}(\text{ARRAY}, \text{DIM}) \leq s_{DIM} + sh \leq \text{UBOUND}(\text{ARRAY}, \text{DIM}) \) holds and is otherwise BOUNDARY or BOUNDARY\((s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_n)\).

**Examples**

Case 1

If \( V \) is the array \([1, 2, 3, 4, 5, 6]\), the effect of shifting \( V \) end-off to the left by 3 positions is achieved by EOSHIFT\((V, \text{SHIFT} = 3)\) which has the value \([4, 5, 6, 0, 0, 0]\).
EOSHIFT(V, SHIFT = -2, BOUNDARY = 99) achieves an end-off shift to the right by 2 positions with the boundary value of 99 and has the value [99, 99, 1, 2, 3, 4].

Case 2

The rows of an array of rank two may all be shifted by the same amount or by different amounts and the boundary elements can be the same or different. If M is the array

\[
\begin{bmatrix}
A & B & C \\
D & E & F \\
G & H & I
\end{bmatrix}
\]

then the value of EOSHIFT(M, SHIFT = -1, BOUNDARY = '*', DIM = 2) is

\[
\begin{bmatrix}
* & A & B \\
* & D & E \\
* & G & H
\end{bmatrix}
\]

and the value of EOSHIFT(M, SHIFT = (/ -1, 1, 0 /), BOUNDARY = (/ '*', '/', '?' /), DIM = 2) is

\[
\begin{bmatrix}
* & A & B \\
E & F & / \\
G & H & I
\end{bmatrix}
\]

**EPSILON(X)**

**Description**

Returns a positive model number that is almost negligible compared to unity in the model representing numbers of the same type and kind type parameter as the argument.
Class
Inquiry function.

Argument
X must be of type real. It may be scalar or array valued.

Result Type, Type Parameter, and Shape
Scalar of the same type and kind type parameter as X.

Result Value
The result has the value $b^p$ where $b$ and $p$ are as defined in the section “The Real Number System Model” for the model representing numbers of the same type and kind type parameter as X.

Examples
EPSILON(X) has the value $2^{-23}$ for real X whose model is described in the section “The Real Number System Model”.
EPSILON(Y), where Y has kind parameter HIGH, would be $2^{-52}$ if $p$ is 48 for the model of kind HIGH.

EXP(X)

Description
Exponential.

Class
Elemental function.

Argument
X must be of type real or complex.
Result Type and Type Parameter
Same as x.

Result Value
The result has a value equal to a processor-dependent approximation to $e^x$. If x is of type complex, its imaginary part is regarded as a value in radians.

Examples
EXP(1.0) has the value 2.7182818.
EXP(2.0_HIGH/3.0) has the value 1.9477340410547 with kind HIGH.

EXPONENT(X)

Description
Returns the exponent part of the argument when represented as a model number.

Class
Elemental function.

Argument
x must be of type real.

Result Type
Default integer.
Result Value
The result has a value equal to the exponent e of the model representation (see the section “The Real Number System Model”) for the value of x, provided x is nonzero and e is within the range for default integers. The result is undefined if the processor cannot represent e in the default integer type. EXPONENT (X) has the value zero if X is zero.

Examples
EXPONENT (1.0) has the value 1 and EXPONENT (4.1) has the value 3 for reals, whose model is described in the section “The Real Number System Model”.

FLOOR(A)

Description
Returns the greatest integer less than or equal to its argument.

Class
Elemental function.

result = CEILING (A, KIND)

Argument
A Input. Must be of type real.
KIND (optional) Input. Must be a scalar integer initialization expression; a Fortran 95 feature.

Result Type and Type Parameter
Default integer.
Result Value

If \texttt{KIND} is present, the kind parameter of the result is that specified by \texttt{KIND}. Otherwise, the kind parameter of the result is that of default integer. The result has a value equal to the least integer greater than or equal to \( A \).

The result is undefined if the processor cannot represent this value in the default integer type.

The result has a value equal to the greatest integer less than or equal to \( A \).

Examples

\begin{itemize}
\item \texttt{FLOOR(3.7)} has the value 3.
\item \texttt{FLOOR(-3.7)} has the value \(-4\).
\item \texttt{FLOOR(10.0\_HIGH/3)} has the value 3.
\end{itemize}

\section*{FRACTION(X)}

\begin{description}
\item[Description] Returns the fractional part of the model representation of the argument value.
\item[Class] Elemental function.
\item[Argument] \( X \) must be of type real.
\item[Result Type and Type Parameter] Same as \( X \).
\end{description}
Result Value
The result has the value $x \times b^{-e}$, where $b$ and $e$ are as defined in the section “The Real Number System Model”. If $x$ has the value zero, the result has the value zero.

Example
`FRACTION(3.0)` has the value 0.75 for reals, whose model is described in “The Real Number System Model”.

---

FREE(A)

---

Description
Frees a block of memory that is currently allocated.

Class
Elemental nonstandard subroutine.

Argument
A must be of type `INTEGER(4)` for IA-32 systems or `INTEGER(8)` for Itanium®-based systems. This value is the starting address of the memory to be freed, previously allocated by `MALLOC`.

If the freed address was not previously allocated by `MALLOC(I)`, or if an address is freed more than once, results are unpredictable.

Example
```
INTEGER(4) ADDR, SIZE.
SIZE = 1024         ! Size in bytes
ADDR = MALLOC(SIZE) ! Allocate the memory
CALL FREE(ADDR)    ! Free it
END
```
HFIX(A)

Description
Convert to INTEGER(2) type.

Class
Elemental nonstandard function.

Argument
A must be of type integer, real, double precision, or complex.

Result
INTEGER(2) type.

Examples
HFIX(9.897) is 9 of type INTEGER(2).
HFIX(9.125) is 9 of type INTEGER(2).
The following code sets b to 34:

```fortran
INTEGER(2) b
COMPLEX p
p = (34.5, 1.0)
b = HFIX(p)
```
**HUGE(X)**

**Description**
Returns the largest number in the model representing numbers of the same type and kind type parameter as the argument.

**Class**
Inquiry function.

**Argument**

\( X \) must be of type integer or real. It may be scalar or array valued.

**Result Type, Type Parameter, and Shape.** Scalar of the same type and kind type parameter as \( X \).

**Result Value**
The result has the value \( r^q \cdot 1 \) if \( X \) is of type integer and
\[
(1 \cdot b^{\lfloor \log_b r \rfloor}) \cdot b^{e_{\text{max}}}
\]
if \( X \) is of type real, where \( r, q, b, p, \) and \( e_{\text{max}} \) are as defined in the section “The Real Number System Model”.

**Example**

\( \text{HUGE}(X) \) has the value \((1 - 2^{-24}) \times 2^{127}\) for real \( X \), whose model is described in “The Real Number System Model”.
IABS(A)

Returns the absolute value of an integer expression

Description
IABS returns the absolute value of an INTEGER*2 expression.

Class
Elemental function.

Argument
A must be of type INTEGER*2 value or expression.

Result Value
If A is positive or zero, the value of A is returned. If A is less than zero, the opposite positive value of A is returned.

Output
Absolute value of A.

IACHAR(C)

Description
Returns the position of a character in the ASCII collating sequence.
Class
Elemental function.

Argument
C must be of type default character and of length one.

Result Type and Type Parameter
Default integer.

Result Value
If C is in the collating sequence defined by the codes specified in ISO 646:1983 (“Information technology — ISO 7-bit coded character set for information interchange”), the result is the position of C in that sequence and satisfies the inequality (0 ≤ IACHAR(C) ≤ 127).

A processor-dependent value is returned if C is not in the ASCII collating sequence. The results are consistent with the LGE, LGT, LLE, and LLT lexical comparison functions. For example, if LLE(C, D) is .TRUE., IACHAR(C) .LE. IACHAR(D) is .TRUE. where C and D are any two characters representable by the processor.

Examples
IACHAR(‘X’) has the value 88.
IACHAR(‘*’) has the value 42.

IADDR(X)

Description
Return the address of X.
**Class**
Inquiry nonstandard function.

**Argument**
X may be of any type.

**Result Type.** The result is of type default integer.
See the section “BADDRESS(X)” for examples.

---

**IAND(I, J)**

**Description**
Performs a bitwise logical AND.

**Class**
Elemental function.

**Argument**
I must be of type integer.
J must be of type integer with the same kind type parameter as I.

**Result Type and Type Parameter**
Same as I.
Intrinsic Procedures

Result Value
The result has the value obtained by combining $I$ and $J$ bit-by-bit according to the following truth table:

<table>
<thead>
<tr>
<th>$I$</th>
<th>$J$</th>
<th>$\text{IAND}(I, J)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The model for the interpretation of an integer value as a sequence of bits is in “The Bit Model”.

Examples
$\text{IAND}(1, 3)$ has the value 1.
$\text{IAND}(2\_\text{SHORT}, 10\_\text{SHORT})$ is 2 with kind SHORT.

IBCLR($I$, $POS$)

Description
Clears a bit to zero.

Class
Elemental function.

Argument
$I$ must be of type integer.
$POS$ must be of type integer. It must be nonnegative and less than BIT_SIZE($I$).
Result Type and Type Parameter
Same as I.

Result Value
The result has the value of the sequence of bits of I, except that bit POS of I is set to zero. The model for the interpretation of an integer value as a sequence of bits is in “The Bit Model”.

Examples
IBCLR(14, 1) has the result 12.
If V has the value (1, 2, 3, 4), the value of IBCLR(POS = V, I = 31) is [29, 27, 23, 15].
The value of IBCLR((/ 15_SHORT, 31_SHORT, 7_SHORT /), 3) is [7, 23, 7] with kind SHORT.

IBITS(I, POS, LEN)

Description
Extracts a sequence of bits.

Class
Elemental function.

Argument
\begin{align*}
I & \quad \text{must be of type integer.} \\
\text{POS} & \quad \text{must be of type integer. It must be nonnegative and POS + LEN must be less than or equal to BIT_SIZE(I).} \\
\text{LEN} & \quad \text{must be of type integer and nonnegative.}
\end{align*}
Result Type and Type Parameter
Same as I.

Result Value
The result has the value of the sequence of LEN bits in I beginning at bit POS, right-adjusted and with all other bits zero. The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

Examples
IBITS(14, 1, 3) has the value 7.
The value of IBITS((/ 15_SHORT, 31_SHORT, 7_SHORT /), 
2_SHORT, 3_SHORT) is [3, 7, 1] with kind SHORT.

IBSET(I, POS)

Description
Sets a bit to one.

Class
Elemental function.

Argument
I must be of type integer.
POS must be of type integer. It must be nonnegative and less than BIT_SIZE(I).

Result Type and Type Parameter
Same as I.
Result Value
The result has the value of the sequence of bits of I, except that bit POS of I is set to one. The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

Examples
IBSET(12, 1) has the value 14.
If V has the value [1, 2, 3, 4], the value of IBSET(POS = V, I = 0) is [2, 4, 8, 16].
The value of IBSET((/ 15_SHORT, 31_SHORT, 7_SHORT /), 3) is [15, 31, 15] with kind SHORT.

ICHAR(C)

Description
Returns the position of a character in the processor collating sequence associated with the kind type parameter of the character.

Class
Elemental function.

Argument
C must be of type character and of length one. Its value must be that of a character capable of representation in the processor.

Result Type and Type Parameter
Default integer.
Result Value

The result is the position of \( C \) in the processor collating sequence associated with the kind type parameter of \( C \) and is in the range \( 0 \leq \text{ICHAR}(C) \leq n-1 \), where \( n \) is the number of characters in the collating sequence.

For any characters \( C \) and \( D \) capable of representation in the processor, 

\[ C \leq D \text{ is .TRUE. if and only if } \text{ICHAR}(C) \leq \text{ICHAR}(D) \text{ is .TRUE.}, \]

and 

\[ C = D \text{ is .TRUE. if and only if } \text{ICHAR}(C) = \text{ICHAR}(D) \text{ is .TRUE.}. \]

Examples

\( \text{ICHAR('X')} \) has the value 88 on a processor using the ASCII collating sequence for the default character type.

\( \text{ICHAR('*')} \) has the value 42 on such a processor.

\[ \text{IDIM(X, Y)} \]

Description

Integer positive difference.

Class

Nonstandard function.

Argument

\( X \) must be of type integer.

\( Y \) must be of type integer with the same kind type parameter as \( X \).

Result Type and Type Parameter

Integer of same kind type parameter as \( X \).
**Result Value**
If \( X > Y \), \( \text{IDIM}(X, Y) \) is \( X - Y \). If \( X \leq Y \), \( \text{IDIM}(X, Y) \) is zero.

**Examples**
- \( \text{IDIM}(89, 12) \) is 77.
- \( \text{IDIM}(56, 59) \) is 0.

---

**IEOR(I, J)**

**Description**
Performs a bitwise exclusive OR.

**Class**
Elemental function.

**Argument**
- \( I \) must be of type integer.
- \( J \) must be of type integer with the same kind type parameter as \( I \).

**Result Type and Type Parameter**
Same as \( I \).
### Result Value

The result has the value obtained by combining $I$ and $J$ bit-by-bit according to the following truth table:

<table>
<thead>
<tr>
<th>$I$</th>
<th>$J$</th>
<th>$\text{IEOR}(I, J)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

### Examples

$\text{IEOR}(1, 3)$ has the value 2.

$\text{IEOR}(\langle 3\_\text{SHORT}, 10\_\text{SHORT} \rangle, 2\_\text{SHORT})$ is $[1, 8]$ with kind $\text{SHORT}$.

---

### IJINT(A)

#### Description

Convert to $\text{INTEGER}(2)$ type.

#### Class

Elemental nonstandard function.

#### Argument

A must be of type $\text{INTEGER}(4)$.
**Result**

INTEGER(2) type.

**Example**

IJINT(32) is 32 of type INTEGER(2).

---

**IMAG(A)**

---

**Description**

Imaginary part of complex number.

**Class**

Elemental nonstandard function.

**Argument**

A must be of type complex or double complex.

**Result**

Real if A is complex. Double precision if A is double complex.

**Example**

The following code sets x to 2.0:

```fortran
COMPLEX p
REAL x
p = (39.61, 2.0)
x = IMAG(p)
```
INDEX(STRING, SUBSTRING, BACK)

Optional Argument

BACK

Description

Returns the starting position of a substring within a string.

Class

Elemental function.

Argument

STRING must be of type character.
SUBSTRING must be of type character with the same kind type parameter as STRING.
BACK (optional) must be of type logical.

Result Type and Type Parameter

Default integer.

Result Value

Case 1 If BACK is absent or present with the value .FALSE., the result is the minimum positive value of I such that
STRING(I : I + LEN(SUBSTRING) - 1) = SUBSTRING or zero if there is no such value.

Zero is returned if LEN(STRING) < LEN(SUBSTRING) and one is returned if LEN(SUBSTRING) = 0.
Case 2 If BACK is present with the value .TRUE., the result is the maximum value of I less than or equal to 
LEN(STRING) – LEN(SUBSTRING) + 1 such that 
STRING(I : I + LEN(SUBSTRING) – 1) = 
SUBSTRING or zero if there is no such value.

Zero is returned if LEN(STRING) < LEN(SUBSTRING) 
and LEN(STRING) + 1 is returned if 
LEN(SUBSTRING) = 0.

Examples
INDEX('FORTRAN', 'R') has the value 3.
INDEX('FORTRAN', 'R', BACK = .TRUE.) has the value 5.
INDEX("XXX", "") has the value 1.
INDEX("XXX", ",", BACK=.TRUE.) has the value 4.

INT(A, KIND)

Optional Argument
KIND

Description
Convert to integer type.

Class
Elemental function.

Argument
A must be of type integer, real, or complex.
KIND (optional) must be a scalar integer initialization expression.
Result Type and Type Parameter

Integer. If KIND is present, the kind type parameter is that specified by KIND; otherwise, the kind type parameter is that of default integer type.

Result Value

Case 1  If A is of type integer, INT(A) = A.

Case 2  If A is of type real, there are two cases: if |A| < 1, INT(A) has the value 0; if |A| ≥ 1, INT(A) is the integer whose magnitude is the largest integer that does not exceed the magnitude of A and whose sign is the same as the sign of A.

Case 3  If A is of type complex, INT(A) is the value obtained by applying the above rules (for reals) to the real part of A. The result is undefined if the processor cannot represent the result in the specified integer type.

Examples

INT(-3.7) has the value –3.

INT(9.1_HIGH/4.0_HIGH, SHORT) is 2 with kind SHORT.

INT1(A)

Description
Convert to INTEGER(1) type.

Class
Elemental nonstandard function.
ARGUMENT

**A** must be of type integer, real, or complex.

RESULT

INTEGER(1) type. If **A** is complex, \texttt{INT1(A)} is equal to the truncated real portion of **A**.

EXAMPLE

\texttt{INT1(6.23)} is 6 of type INTEGER(1).

INT2(A)

**Description**

Convert to INTEGER(2) type.

**Class**

Elemental nonstandard function.

**Argument**

**A** must be of type integer, real, or complex.

**Result**

INTEGER(2) type. If **A** is complex, \texttt{INT2(A)} is equal to the truncated real portion of **A**.

**Example**

\texttt{INT2(212.4545)} is 212 of type INTEGER(2).
**INT4(A)**

**Description**
Convert to INTEGER(4) type.

**Class**
Elemental nonstandard function.

**Argument**
A must be of type integer, real, or complex.

**Result**
INTEGER(4) type. If A is complex, INT4(A) is equal to the truncated real portion of A.

**Example**

---

**INT8(A)**

**Description**
Convert to INTEGER(8) type.

**Class**
Elemental nonstandard function.
**Argument**

A must be of type integer, real, or complex.

**Result**

INTEGER(8) type. If A is complex, INT8(A) is equal to the truncated real portion of A.

**Example**

INT8(14.14) is 14 of type INTEGER(8).

---

**INUM(I)**

**Description**

Convert character to INTEGER(2) type.

**Class**

Elemental nonstandard function.

**Argument**

I must be of type character.

**Result**

INTEGER(2) type.

**Example**

INUM("451.92") is 451 of type INTEGER(2).
**IOR(I, J)**

**Description**
Performs a bitwise inclusive OR.

**Class**
Elemental function.

**Argument**
- **I** must be of type integer.
- **J** must be of type integer with the same kind type parameter as I.

**Result Type and Type Parameter**
Same as I.

**Result Value**
The result has the value obtained by combining I and J bit-by-bit according to the following truth table:

<table>
<thead>
<tr>
<th>I</th>
<th>J</th>
<th>IOR(I, J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

The Bit Model
IQINT(A)

**Description**
Convert to integer type.

**Class**
Elemental nonstandard function.

**Argument**
A must be of type REAL(16).

**Result**
Integer type.

**Examples**
IQINT(9416.39) is 9416.

ISHFT(I, SHIFT)

**Description**
Performs a logical shift.

**Class**
Elemental function.
**Argument**

I  
must be of type integer.

SHIFT  
must be of type integer. The absolute value of SHIFT must be less than or equal to BIT_SIZE(I).

**Result Type and Type Parameter**

Same as I.

**Result Value**

The result has the value obtained by shifting the bits of I by SHIFT positions.

If SHIFT is positive, the shift is to the left; if SHIFT is negative, the shift is to the right; and if SHIFT is zero, no shift is performed. Bits shifted out from the left or from the right, as appropriate, are lost. Zeros are shifted in from the opposite end.

The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

**Examples**

ISHFT(3, 1) has the value 6.
ISHFT(3, -1) has the value 1.

---

**ISHFTC(I, SHIFT, SIZE)**

---

**Optional Argument**

SIZE

**Description**

Performs a circular shift of the rightmost bits.
Class
Elemental function.

Argument
I must be of type integer.
SHIFT must be of type integer. The absolute value of SHIFT must be less than or equal to SIZE.
SIZE (optional) must be of type integer. The value of SIZE must be positive and must not exceed BIT_SIZE(I). If SIZE is absent, it is as if it were present with the value of BIT_SIZE(I).

Result Type and Type Parameter
Same as I.

Result Value
The result has the value obtained by shifting the SIZE rightmost bits of I circularly by SHIFT positions.
If SHIFT is positive, the shift is to the left; if SHIFT is negative, the shift is to the right; and if SHIFT is zero, no shift is performed. No bits are lost. The unshifted bits are unaltered.
The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

Example
ISHFTC(3, 2, 3) has the value 5.
ISIGN(A, B)

Description
Absolute value of A times the sign of B.

Class
Elemental nonstandard function.

Argument
A must be of type integer.
B must be of type integer with the same kind type parameter as A.

Result Type and Type Parameter
Same as A.

Result Value
The value of the result is |A| if B ≥ 0 and -|A| if B < 0.

Examples
ISIGN(-3, 0) is 3.
ISIGN(12, -9) is -12.
**ISNAN(X)**

**Description**
Determine if a value is NaN (not a number).

**Class**
Elemental nonstandard function.

**Argument**
X must be of type real.

**Result Type**
Logical.

**Examples**
ISNAN(45.4) is .FALSE.
ISNAN(ACOSH(0.0)) is .TRUE.

---

**IXOR(I, J)**

**Description**
Exclusive OR.

**Class**
Elemental nonstandard function.
Intrinsic Procedures

Argument

I must be of type integer.
J must be of type integer with the same kind type parameter as I.

Result Type and Type Parameter

Same as I.

Result Value

The result has the value obtained by performing an exclusive OR on I and J bit-by-bit according to the truth table that follows.

<table>
<thead>
<tr>
<th>I</th>
<th>J</th>
<th>IXOR (I, J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The model for interpreting an integer value as a sequence of bits is in the section “The Bit Model”.

Example

IXOR (12, 7) is 11. (Binary 1100 exclusive OR with binary 0111 is binary 1011.)
JNUM(I)

Description
Convert character to integer type.

Class
Elemental nonstandard function.

Argument
I must be of type character.

Result
Integer type.

Example
JNUM("46616.725") is 46616.

KIND(X)

Description
Returns the value of the kind type parameter of X.

Class
Inquiry function.
**Argument**

X may be of any intrinsic type.

**Result Type, Type Parameter, and Shape**

Default integer scalar.

**Result Value**

The result has a value equal to the kind type parameter value of X.

**Examples**

KIND (0.0) has the kind type parameter value of default real.

KIND (1.0_HIGH) has the value of the named constant HIGH.

---

**LBOUND(ARRAY, DIM)**

---

**Optional Argument**

DIM

**Description**

Returns all the lower bounds or a specified lower bound of an array.

**Class**

Inquiry function.

**Argument**

ARRAY may be of any type. It must not be scalar. It must not be a pointer that is disassociated or an allocatable array that is not allocated.
DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.

Result Type, Type Parameter, and Shape
The result is of type default integer. It is scalar if DIM is present; otherwise, the result is an array of rank one and size $n$, where $n$ is the rank of ARRAY.

Result Value
Case 1 For an array section or for an array expression other than a whole array or array structure component, LBOUND (ARRAY, DIM) has the value 1. For a whole array or array structure component, LBOUND (ARRAY, DIM) has the value:
- equal to the lower bound for subscript DIM of ARRAY if dimension DIM of ARRAY does not have extent zero or if ARRAY is an assumed-size array of rank DIM
or
- one (1), otherwise.

Case 2 LBOUND (ARRAY) has a value whose $i$th component is equal to LBOUND (ARRAY, $i$), for $i = 1, 2, ..., n$, where $n$ is the rank of ARRAY.

Examples
If the following statements are processed

REAL, TARGET :: A (2:3, 7:10)
REAL, POINTER, DIMENSION (:,:) :: B, C, D
B => A
C => A(:, :)
ALLOCATE ( D(-3:3,-7:7) )

LBOUND (A) is [2, 7], LBOUND (A, DIM=2) is 7, LBOUND (B) is [2,7], LBOUND (C) is [1,1], and LBOUND (D) is [-3,-7].
LEN(STRING)

Description
Returns the length of a character entity.

Class
Inquiry function.

Argument
STRING must be of type character. It may be scalar or array valued.

Result Type, Type Parameter, and Shape
Default integer scalar.

Result Value
The result has a value equal to the number of characters in STRING if it is scalar or in an element of STRING if it is array valued.

Example
If C and D are declared by the statements
CHARACTER (11) C(100)
CHARACTER (LEN=31) D
LEN(C) has the value 11, and LEN(D) has the value 31.
LEN_TRIM(STRING)

**Description**
Returns the length of the character argument without counting trailing blank characters.

**Class**
Elemental function.

**Argument**
STRING must be of type character.

**Result Type and Type Parameter**
Default integer.

**Result Value**
The result has a value equal to the number of characters remaining after any trailing blanks in STRING are removed. If the argument contains no nonblank characters, the result is zero.

**Examples**
LEN_TRIM(’ bA bb b ’) has the value 4.
LEN_TRIM(’ bbb ’) has the value 0.
LGE(STRING_A, STRING_B)

Description
Tests whether a string is lexically greater than or equal to another string, based on the ASCII collating sequence.

Class
Elemental function.

Argument
STRING_A must be of type default character.
STRING_B must be of type default character.

Result Type and Type Parameters
Default logical.

Result Value
If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string.

If either string contains a character not in the ASCII character set, the result is processor dependent.

The result is .TRUE. if the strings are equal or if STRING_A follows STRING_B in the ASCII collating sequence; otherwise, the result is .FALSE.. Note that the result is .TRUE. if both STRING_A and STRING_B are of zero length.

Examples
LGE(‘apple’, ‘beans’) has the value .FALSE..
LGE(‘apple’, ‘applesauce’) has the value .FALSE..
LGE(‘Zebra’, ‘Yak’) has the value .TRUE..
LGT(STRING_A, STRING_B)

Description
Tests whether a string is lexically greater than another string, based on the ASCII collating sequence.

Class
Elemental function.

Argument
STRING_A must be of type default character.
STRING_B must be of type default character.

Result Type and Type Parameters
Default logical.

Result Value
If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string.
If either string contains a character not in the ASCII character set, the result is processor-dependent.
The result is .TRUE. if STRING_A follows STRING_B in the ASCII collating sequence; otherwise, the result is .FALSE.. Note that the result is .FALSE. if both STRING_A and STRING_B are of zero length.

Examples
LGT(‘apple’, ‘beans’) has the value .FALSE..
LGT(‘apple’, ‘applesauce’) has the value .FALSE..
LGT(‘Zebra’, ‘Yak’) has the value .TRUE.
LLE(STRING_A, STRING_B)

Description
Tests whether a string is lexically less than or equal to another string, based on the ASCII collating sequence.

Class
Elemental function.

Argument
STRING_A must be of type default character.
STRING_B must be of type default character.

Result Type and Type Parameters
Default logical.

Result Value
If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string.
If either string contains a character not in the ASCII character set, the result is processor dependent.
The result is .TRUE. if the strings are equal or if STRING_A precedes STRING_B in the ASCII collating sequence; otherwise, the result is .FALSE.. Note that the result is .TRUE. if both STRING_A and STRING_B are of zero length.

Examples
LLE(‘apple’, ‘beans’) has the value .TRUE..
LLE(‘apple’, ‘applesauce’) has the value .TRUE..
LLE(‘Zebra’, ‘Yak’) has the value .FALSE..
**LLT(STRING_A, STRING_B)**

**Description**
Tests whether a string is lexically less than another string, based on the ASCII collating sequence.

**Class**
Elemental function.

**Argument**
- STRING_A must be of type default character.
- STRING_B must be of type default character.

**Result Type and Type Parameters**
Default logical.

**Result Value**
If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string.

If either string contains a character not in the ASCII character set, the result is processor-dependent.

The result is .TRUE. if STRING_A precedes STRING_B in the ASCII collating sequence; otherwise, the result is .FALSE.. Note that the result is .FALSE. if both STRING_A and STRING_B are of zero length.

**Examples**
- LLT(‘apple’, ‘beans’) has the value .TRUE..
- LLT(‘apple’, ‘applesauce’) has the value .TRUE..
- LLT(‘Zebra’, ‘Yak’) has the value .FALSE..
**LOC(X)**

**Description**
Return the address of the argument.

**Class**
Inquiry nonstandard function.
For details see “LOC” on page 86.

**LOG(X)**

**Description**
Natural logarithm.

**Class**
Elemental function.

**Argument**
x must be of type real or complex. If x is real, its value must be greater than zero. If x is complex, its value must not be zero.

**Result Type and Type Parameter**
Same as x.
Result Value
The result has a value equal to a processor-dependent approximation to loge x. A result of type complex is the principal value with imaginary part ω in the range -π < ω ≤ π. The imaginary part of the result is π only when the real part of the argument is less than zero and the imaginary part of the argument is zero.

Examples
\[
\log(10.0) \text{ has the value } 2.3025851.
\]
\[
\log((-0.5 \_H I G H, 0)) \text{ has the value:}
\]
\[
-0.69314718055994 + 3.1415926535898i \text{ with kind } \text{HIGH}.
\]

LOG10(X)

Description
Common logarithm.

Class
Elemental function.

Argument
x must be of type real. The value of x must be greater than zero.

Result Type and Type Parameter
Same as x.
Result Value
The result has a value equal to a processor-dependent approximation to $\log_{10}X$.

Examples
\[
\text{LOG10}(10.0) \text{ has the value 1.0.}
\]
\[
\text{LOG10}(10.0\text{E1000\_HIGH}) \text{ has the value 1001.0 with kind\_HIGH.}
\]

LOGICAL(L, KIND)

Optional Argument
KIND

Description
Converts between kinds of logical.

Class
Elemental function.

Argument
L must be of type logical.
KIND (optional) must be a scalar integer initialization expression.

Result Type and Type Parameter
Logical. If KIND is present, the kind type parameter is that specified by KIND; otherwise, the kind type parameter is that of default logical.

Result Value
The value is that of L.
Examples

LOGICAL(L .OR. .NOT. L) has the value TRUE. and is of type default logical, regardless of the kind type parameter of the logical variable L.

LOGICAL(L, BIT) has kind parameter BIT and has the same value as L.

### LSHIFT(I, SHIFT)

**Description**
Left shift.

**Class**
Elemental nonstandard function.
For details see “LSHFT” on page 90.

### LSHIFT(I, SHIFT)

**Description**
Left shift.

**Class**
Elemental nonstandard function.
For details see “LSHFT” on page 91.
MALLOC (I)

Description
Allocates a block of memory. This nonstandard function has no generic function associated with it. It must not be passed as an actual argument.

Class
Elemental nonstandard function.

Argument
I must be of type INTEGER(4). This value is the size (in bytes) of memory to be allocated.

Result
The result type is INTEGER(4) for IA-32 systems or INTEGER(8) for Itanium®-based systems. The result is the starting address of the allocated memory. The memory allocated can be freed by using the FREE(A) intrinsic function.

Example
INTEGER(4) ADDR, SIZE.
SIZE = 1024 ! Size in bytes
ADDR = MALLOC(SIZE) ! Allocate the memory
CALL FREE(ADDR) ! Free it
END
MATMUL(MATRIX_A, MATRIX_B)

Description
Performs matrix multiplication of numeric or logical matrices.

Class
Transformational function.

Argument
- **MATRIX_A**: must be of numeric type (integer, real, or complex) or of logical type. It must be array valued and of rank one or two.
- **MATRIX_B**: must be of numeric type if **MATRIX_A** is of numeric type and of logical type if **MATRIX_A** is of logical type. It must be array valued and of rank one or two.

If **MATRIX_A** has rank one, **MATRIX_B** must have rank two. If **MATRIX_B** has rank one, **MATRIX_A** must have rank two. The size of the first (or only) dimension of **MATRIX_B** must equal the size of the last (or only) dimension of **MATRIX_A**.

Result Type, Type Parameter, and Shape
If the arguments are of numeric type, the type and kind type parameter of the result are determined by the types of **MATRIX_A** and **MATRIX_B**.
If the arguments are of type logical, the result is of type logical with the kind type parameter of the arguments.

The shape of the result depends on the shapes of the arguments as follows:

- **Case 1**: If **MATRIX_A** has shape \([n, m]\) and **MATRIX_B** has shape \([m, k]\), the result has shape \([n, k]\).
- **Case 2**: If **MATRIX_A** has shape \([m]\) and **MATRIX_B** has shape \([m, k]\), the result has shape \([k]\).
Case 3  If \( \text{MATRIX}_A \) has shape \([n, m]\) and \( \text{MATRIX}_B \) has shape \([m]\), the result has shape \([n]\).

**Result Value**

Case 1  Element \((i, j)\) of the result has the value
\[
\text{SUM}(\text{MATRIX}_A(i, :) \ast \text{MATRIX}_B(:, j))
\]
if the arguments are of numeric type and has the value
\[
\text{ANY}(\text{MATRIX}_A(i, :) \, \&\, \& \, \text{MATRIX}_B(:, j))
\]
if the arguments are of logical type.

Case 2  Element \((j)\) of the result has the value
\[
\text{SUM}(\text{MATRIX}_A(:, :) \ast \text{MATRIX}_B(:, j))
\]
if the arguments are of numeric type and has the value
\[
\text{ANY}(\text{MATRIX}_A(:, :) \, \&\, \& \, \text{MATRIX}_B(:, j))
\]
if the arguments are of logical type.

Case 3  Element \((i)\) of the result has the value
\[
\text{SUM}(\text{MATRIX}_A(i, :) \ast \text{MATRIX}_B(:))
\]
if the arguments are of numeric type and has the value
\[
\text{ANY}(\text{MATRIX}_A(i, :) \, \&\, \& \, \text{MATRIX}_B(:))
\]
if the arguments are of logical type.

**Examples**

If \( A \) is the matrix
\[
\begin{bmatrix}
1 & 2 & 3 \\
2 & 3 & 4
\end{bmatrix}
\]
and \( B \) is the matrix
\[
\begin{bmatrix}
1 & 2 \\
2 & 3 \\
3 & 4
\end{bmatrix}
\]
and \( X \) is the vector \([1, 2]\) and \( Y \) is the vector \([1, 2, 3]\).

Case 1  The result of \( \text{MATMUL}(A, B) \) is the matrix-matrix product \( AB \) with the value
\[
\begin{bmatrix}
123 \\
234
\end{bmatrix}
\]
Case 2  The result of \texttt{MATMUL(X, A)} is the vector-matrix product $XA$ with the value $[5, 8, 11]$.

Case 3  The result of \texttt{MATMUL(A, Y)} is the matrix-vector product $AY$ with the value $[14, 20]$.

\section*{MAX(A1, A2, A3, ...)}

\begin{itemize}
\item \textbf{Optional Arguments}
\begin{itemize}
\item $A3$, ...
\end{itemize}
\item \textbf{Description}
\text{Maximum value.}
\item \textbf{Class}
\text{Elemental function.}
\item \textbf{Arguments}
The arguments must all have the same type which must be integer or real, and they must all have the same kind type parameter.
\item \textbf{Result Type and Type Parameter}
Same as the arguments.
\item \textbf{Result Value}
The value of the result is that of the largest argument.
\end{itemize}
Examples
MAX(-9.0, 7.0, 2.0) has the value 7.0.
MAX(-1.0_HIGH/3, -0.1_HIGH) is -0.1_HIGH.

MAXEXPONENT(X)

Description
Returns the maximum exponent in the model representing numbers of the same type and kind type parameter as the argument.

Class
Inquiry function.

Argument
X must be of type real. It may be scalar or array valued.

Result Type, Type Parameter, and Shape
Default integer scalar.

Result Value
The result has the value $e_{\text{max}}$, as defined in the section “The Real Number System Model”.

Example
MAXEXPONENT(X) has the value 128 for real X, whose model is described in the section “The Real Number System Model”.
MAXLOC(ARRAY, DIM, MASK, KIND)

Optional ArgumentS
DIM, MASK, KIND

Description
Returns the location of the maximum value of all elements in an array, a set of elements in an array, or elements in a specified dimension of an array having the maximum value of the elements identified by MASK.

Class
Transformational function.

Arguments
| ARRAY | Input; must be of type integer or real. It must not be scalar. |
| DIM (optional) | Input; must be a scalar integer with a value in the range 1 to n, where n is the rank of ARRAY. This argument is a Fortran 95 feature. |
| MASK (optional) | Input; must be of type logical and must be conformable with ARRAY. |
| KIND (optional) | Input; must be a scalar integer initialization |

Result Type, Type Parameter, and Shape
The result is of type default integer; it is an array of rank one and of size equal to the rank of ARRAY.
Result Value

Case 1  If \texttt{MASK} is absent, the result is a rank-one array whose element values are the values of the subscripts of an element of \texttt{ARRAY} whose value equals the maximum value of all of the elements of \texttt{ARRAY}.

The \(i\)th subscript returned lies in the range 1 to \(e_i\), where \(e_i\) is the extent of the \(i\)th dimension of \texttt{ARRAY}.

If more than one element has the maximum value, the element whose subscripts are returned is the first such element, taken in array element order. If \texttt{ARRAY} has size zero, the value of the result is processor-dependent.

Case 2  If \texttt{MASK} is present, the result is a rank-one array whose element values are the values of the subscripts of an element of \texttt{ARRAY}, corresponding to a \texttt{.TRUE.} element of \texttt{MASK}, whose value equals the maximum value of all such elements of \texttt{ARRAY}.

The \(i\)th subscript returned lies in the range 1 to \(e_i\), where \(e_i\) is the extent of the \(i\)th dimension of \texttt{ARRAY}.

If more than one such element has the maximum value, the element whose subscripts are returned is the first such element taken in array element order.

If there are no such elements (that is, if \texttt{ARRAY} has size zero or every element of \texttt{MASK} has the value \texttt{.FALSE.}), the value of the result is processor-dependent.

In both cases, an element of the result is undefined if the processor cannot represent the value as a default integer.
Examples
Case 1 The value of MAXLOC((/ 2, 6, 4, 6 /)) is [2].
   If the array B is declared
   INTEGER, DIMENSION(4:7) :: B = (/ 8, 6, 3, 1 /)
   the value of MAXLOC(B) is [1].
Case 2 If A has the value
   0  D5  8  D3
   3  4  D1  2
   1  5  6  D4
   then MAXLOC(A, MASK = A .LT. 6) has the value
   [3, 2]. Note that this is true even if A has a declared
   lower bound other than 1.

MAXVAL(ARRAY, DIM, MASK)

Optional Arguments
DIM, MASK

Description
Maximum value of the elements of ARRAY along dimension DIM that
correspond to the .TRUE. elements of MASK.

Class
Transformational function.
Arguments

**ARRAY**
- must be of type integer or real. It must not be scalar.

**DIM** (optional)
- must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\) where \(n\) is the rank of **ARRAY**. The corresponding actual argument must not be an optional dummy argument.

**MASK** (optional)
- must be of type logical and must be conformable with **ARRAY**.

Result Type, Type Parameter, and Shape

The result is of the same type and kind type parameter as **ARRAY**.

It is scalar if **DIM** is absent or **ARRAY** has rank one; otherwise, the result is an array of rank \(n-1\) and of shape \((d_1, d_2, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)\) where \((d_1, d_2, \ldots, d_n)\) is the shape of **ARRAY**.

Result Value

Case 1
- The result of \(\text{MAXVAL}(\text{ARRAY})\) has a value equal to the maximum value of all the elements of **ARRAY** or has the value of the negative number of the largest magnitude supported by the processor for numbers of the type and kind type parameter of **ARRAY** if **ARRAY** has size zero.

Case 2
- The result of \(\text{MAXVAL}(\text{ARRAY}, \text{MASK} = \text{MASK})\) has a value equal to the maximum value of the elements of **ARRAY** corresponding to `.TRUE.` elements of **MASK** or has the value of the negative number of the largest magnitude supported by the processor for numbers of the same type and kind type parameter as **ARRAY** if there are no `.TRUE.` elements.

Case 3
- If **ARRAY** has rank one, \(\text{MAXVAL}(\text{ARRAY}, \text{DIM} [, \text{MASK}])\) has a value equal to that of \(\text{MAXVAL}(\text{ARRAY} [, \text{MASK}])\). Otherwise, the value of element \((s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)\) of \(\text{MAXVAL}(\text{ARRAY}, \text{DIM} [, \text{MASK}])\) is equal to the following:

\[
\text{MAXVAL}(\text{ARRAY} (s_1, s_2, \ldots, s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, \ldots, s_n) [, \text{MASK} = \text{MASK} (s_1, s_2, \ldots, s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, \ldots, s_n) ])
\]
Examples

Case 1  The value of \texttt{MAXVAL((/ 1, 2, 3 /))} is 3.

Case 2  \texttt{MAXVAL(C, MASK = C .LT. 0.0)} finds the maximum of the negative elements of \texttt{C}.

Case 3  If \texttt{B} is the array

\[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix}
\]

then \texttt{MAXVAL(B, DIM = 1)} is [2, 4, 6] and \texttt{MAXVAL(B, DIM = 2)} is [5, 6].

\section*{MCLOCK()}

### Description

Return time accounting for a program.

### Class

Inquiry nonstandard function.

### Result Type

Integer.

### Result Value

The value returned, in units of microseconds, is the sum of the current process’s user time and the user and system time of all its child processes.
**MERGE(TSOURCE, FSOURCE, MASK)**

**Description**
Choose alternative value according to the value of a mask.

**Class**
Elemental function.

**Arguments**
- **TSOURCE**: may be of any type.
- **FSOURCE**: must be of the same type and type parameters as **TSOURCE**.
- **MASK**: must be of type logical.

**Result Type and Type Parameters**
Same as **TSOURCE**.

**Result Value**
The result is **TSOURCE** if **MASK** is `.TRUE.` and **FSOURCE** otherwise.

**Examples**
If **TSOURCE** is the array
\[
\begin{bmatrix}
1 & 6 & 5 \\
2 & 4 & 6 \\
\end{bmatrix}
\]

and **FSOURCE** is the array
\[
\begin{bmatrix}
0 & 3 & 2 \\
7 & 4 & 8 \\
\end{bmatrix}
\]

and **MASK** is the array
\[
\begin{bmatrix}
\text{T} & \text{T} \\
\text{.} & \text{.} \\
\end{bmatrix}
\]
then where “T” represents .TRUE. and “.” represents .FALSE., then
MERGE(TSOURCE, FSOURCE, MASK) is

\[
\begin{pmatrix}
1 & 3 & 5 \\
7 & 4 & 6
\end{pmatrix}
\]

The value of \(\text{MERGE}(1.0, 0.0, K > 0)\) is 1.0 for \(K = 5\) and 0.0 for \(K = -2\).

---

**MIN(A1, A2, A3, ...)**

---

**Optional Arguments**

A3, ...

**Description**

Minimum value.

**Class**

Elemental function.

**Arguments**

The arguments must all be of the same type, which must be integer or real, and they must all have the same kind type parameter.

**Result Type and Type Parameter**

Same as the arguments.

**Result Value**

The value of the result is that of the smallest argument.
Examples
MIN(-9.0, 7.0, 2.0) has the value -9.0.
MIN(-0.4_HIGH, -1.0_HIGH/3) is -0.4_HIGH.

MINEXPONENT(X)

Description
Returns the minimum exponent in the model representing numbers of the same type and kind type parameter as the argument.

Class
Inquiry function.

Argument
x must be of type real. It may be scalar or array valued.

Result Type, Type Parameter, and Shape
Default integer scalar.

Result Value
The result has the value $e_{\text{min}}$, as defined in the section “The Real Number System Model”.

Example
MINEXPONENT(X) has the value $-125$ for real X, whose model is described in “The Real Number System Model”.
MINLOC(ARRAY, DIM, MASK, KIND)

Optional Argument
DIM, MASK, KIND

Description
Returns the location of the minimum value of all elements in an array, a set of elements in an array, or elements in a specified dimension of an array having the maximum value of the elements identified by MASK.

Class
Transformational function.

Arguments
ARRAY must be of type integer or real. It must not be scalar.
DIM (optional) Input; must be a scalar integer with a value in the range 1 to n, where n is the rank of ARRAY. This argument is a Fortran 95 feature.
MASK (optional) Input; must be of type logical and must be conformable with ARRAY.
KIND (optional) Input; must be a scalar integer initialization.

Result Type, Type Parameter, and Shape
The result is of type default integer; it is an array of rank one and of size equal to the rank of ARRAY.
Result Value

Case 1  If \texttt{MASK} is absent, the result is a rank-one array whose element values are the values of the subscripts of an element of \texttt{ARRAY} whose value equals the minimum value of all the elements of \texttt{ARRAY}.

The \(i\)th subscript returned lies in the range 1 to \(e_i\), where \(e_i\) is the extent of the \(i\)th dimension of \texttt{ARRAY}.

If more than one element has the minimum value, the element whose subscripts are returned is the first such element, taken in array element order. If \texttt{ARRAY} has size zero, the value of the result is processor-dependent.

Case 2  If \texttt{MASK} is present, the result is a rank-one array whose element values are the values of the subscripts of an element of \texttt{ARRAY}, corresponding to a \texttt{.TRUE.} element of \texttt{MASK}, whose value equals the minimum value of all such elements of \texttt{ARRAY}.

The \(i\)th subscript returned lies in the range 1 to \(e_i\), where \(e_i\) is the extent of the \(i\)th dimension of \texttt{ARRAY}. If more than one such element has the minimum value, the element whose subscripts are returned is the first such element taken in array element order.

If \texttt{ARRAY} has size zero or every element of \texttt{MASK} has the value \texttt{.FALSE.}, the value of the result is processor-dependent.

In both cases, an element of the result is undefined if the processor cannot represent the value as a default integer.

Examples

Case 1  The value of \texttt{MINLOC((/ 4, 3, 6, 3 /))} is \[2\].

If the array \texttt{B} is declared

\begin{verbatim}
INTEGER, DIMENSION(4:7) :: B = (/ 8, 6, 3, 1 /)
\end{verbatim}

the value of \texttt{MINLOC(B)} is \[4\]

Case 2  If \texttt{A} has the value
then \( \text{MINLOC}(A, \text{MASK} = A \ .\ GT\ .\ -4) \) has the value \([1, 4]\). Note that this is true even if \( A \) has a declared lower bound other than 1.

**MINVAL(ARRAY, DIM, MASK)**

**Optional Arguments**

DIM, MASK

**Description**

Minimum value of all the elements of ARRAY along dimension DIM corresponding to \( \text{.TRUE.} \) elements of MASK.

**Class**

Transformational function.

**Arguments**

- **ARRAY**
  - Must be of type integer or real. It must not be scalar.
- **DIM (optional)**
  - Must be scalar and of type integer with a value in the range \( 1 \leq \text{DIM} \leq n \), where \( n \) is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.
- **MASK (optional)**
  - Must be of type logical and must be conformable with ARRAY.
Result Type, Type Parameter, and Shape

The result is of the same type and kind type parameter as `ARRAY`. It is scalar if `DIM` is absent or `ARRAY` has rank one; otherwise, the result is an array of rank \( n-1 \) and of shape \((d_1, d_2, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)\) where \((d_1, d_2, \ldots, d_n)\) is the shape of `ARRAY`.

Result Value

Case 1  
The result of \( \text{MINVAL}(\text{ARRAY}) \) has a value equal to the minimum value of all the elements of `ARRAY` or has the value of the positive number of the largest magnitude supported by the processor for numbers of the type and kind type parameter of `ARRAY` if `ARRAY` has size zero.

Case 2  
The result of \( \text{MINVAL}(\text{ARRAY}, \text{MASK} = \text{MASK}) \) has a value equal to the minimum value of the elements of `ARRAY` corresponding to `.TRUE.` elements of `MASK` or has the value of the positive number of the largest magnitude supported by the processor for numbers of the same type and kind type parameter as `ARRAY` if there are no `.TRUE.` elements.

Case 3  
If `ARRAY` has rank one, \( \text{MINVAL}(\text{ARRAY}, \text{DIM} [,\text{MASK}]) \) has a value equal to that of \( \text{MINVAL}(\text{ARRAY} [,\text{MASK} = \text{MASK}]) \). Otherwise, the value of element \((s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)\) of \( \text{MINVAL}(\text{ARRAY}, \text{DIM} [,\text{MASK}]) \) is equal to the following:

\[
\text{MINVAL}(\text{ARRAY}(s_1, s_2, \ldots, s_{\text{DIM}-1}, : , s_{\text{DIM}+1}, \ldots, s_n) [,\text{MASK} = \text{MASK}(s_1, s_2, \ldots, s_{\text{DIM}-1}, : , s_{\text{DIM}+1}, \ldots, s_n)])
\]

Examples

Case 1  
The value of \( \text{MINVAL}((/ \ 1, 2, 3 /)) \) is 1.

Case 2  
\( \text{MINVAL}(C, \text{MASK} = C \ .\text{GT.} \ 0.0) \) forms the minimum of the positive elements of `C`.

Case 3  
If `B` is the array
then \( \text{MINVAL}(B, \ DIM = 1) \) is \([1, 3, 5]\) and \( \text{MINVAL}(B, \ DIM = 2) \) is \([1, 2]\).

### MOD(A, P)

**Description**
Remainder function.

**Class**
Elemental function.

**Arguments**
- \( A \) must be of type integer or real.
- \( P \) must be of the same type and kind type parameter as \( A \).

**Result Type and Type Parameter**
Same as \( A \).

**Result Value**
If \( P \neq 0 \), the value of the result is \( A - \text{INT}(A/P) \times P \). If \( P=0 \), the result is processor-dependent.

**Examples**
- \( \text{MOD}(3.0, 2.0) \) has the value \( 1.0 \).
- \( \text{MOD}(8, 5) \) has the value \( 3 \).
- \( \text{MOD}(-8, 5) \) has the value \( -3 \).
- \( \text{MOD}(8, -5) \) has the value \( 3 \).
MODULO(A, P)

Description
Modulo function.

Class
Elemental function.

Arguments

A  must be of type integer or real.
P  must be of the same type and kind type parameter as A.

Result Type and Type Parameter
Same as A.

Result Value
Case 1  A is of type integer. If P≠0, MODULO (A, P) has the value R such that A = Q × P + R, where Q is an integer, the inequalities 0≤R<P hold if P>0, and P<R≤0 hold if P<0. If P=0, the result is processor-dependent.
Case 2  A is of type real. If P≠0, the value of the result is \( A - \text{FLOOR}(A / P) \times P \). If P=0, the result is processor-dependent.

Examples
MODULO (8, 5) has the value 3.
MODULO (-8, 5) has the value 2.
MODULO(8, -5) has the value –2.
MODULO(-8, -5) has the value –3.
MODULO(3.0, 2.0) has the value 1.0.
MODULO(2.0_HIGH, 3.0_HIGH) has the value 2.0_HIGH.

MVBITS(FROM, FROMPOS, LEN, TO, TOPOS)

**Description**
Copies a sequence of bits from one data object to another.

**Class**
Elemental subroutine.

**Arguments**
- **FROM** must be of type integer. It is an **INTENT(IN)** argument.
- **FROMPOS** must be of type integer and nonnegative. It is an **INTENT(IN)** argument. **FROMPOS + LEN** must be less than or equal to **BIT_SIZE(FROM)**. The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.
- **LEN** must be of type integer and nonnegative. It is an **INTENT(IN)** argument.
- **TO** must be a variable of type integer with the same kind type parameter value as **FROM** and may be the same variable as **FROM**. It is an **INTENT(INOUT)** argument. **TO** is set by copying the sequence of bits of length **LEN**, starting at position **FROMPOS** of **FROM** to position **TOPOS** of **TO**. No other bits of **TO** are altered. On return, the
LEN bits of TO starting at TOPOS are equal to the value that the LEN bits of FROM starting at FROMPOS had on entry.

The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

TOPOS must be of type integer and nonnegative. It is an INTENT (IN) argument. TOPOS + LEN must be less than or equal to BIT_SIZE(TO).

**Examples**

If TO has the initial value 6, the value of TO after the statement CALL MVBITS(7, 2, 2, TO, 0) is 5.

After the statement

```plaintext
CALL MVBITS (PATTERN, 0_SHORT, 1_SHORT,
               PATTERN, 7_SHORT)
```

is executed, the integer variable PATTERN of kind SHORT has a leading bit that is identical to its terminal bit.

---

**NEAREST(X, S)**

**Description**

Returns the nearest different machine representable number in a given direction.

**Class**

Elemental function.

**Arguments**

- \(X\) must be of type real.
S must be of type real and not equal to zero.

**Result Type and Type Parameter**
Same as \( x \).

**Result Value**
The result has a value equal to the machine representable number distinct from \( x \) and nearest to it in the direction of the infinity with the same sign as \( S \).

**Example**
\[
\text{NEAREST}(3.0, 2.0) \text{ has the value } 3+2^{-22} \text{ on a machine whose representation is that of the model described in the section “The Real Number System Model”}.
\]

---

**NINT(A, KIND)**

---

**Optional Argument**

**KIND**

**Description**
Nearest integer.

**Class**
Elemental function.

**Arguments**

A must be of type real.

KIND (optional) must be a scalar integer initialization expression.
Result Type and Type Parameter
Integer. If KIND is present, the kind type parameter is that specified by KIND; otherwise, the kind type parameter is that of default integer type.

Result Value
If $A > 0$, $\text{NINT}(A)$ has the value $\text{INT}(A+0.5)$; if $A \leq 0$, $\text{NINT}(A)$ has the value $\text{INT}(A-0.5)$. The result is undefined if the processor cannot represent the result in the specified integer type.

Examples
NINT(2.783) has the value 3.
NINT(-1.99999999999_HIGH) has the value -2.

NOT(I)

Description
Performs a bitwise logical complement.

Class
Elemental function.

Argument
I must be of type integer.

Result Type and Type Parameter
Same as I.
**Result Value**

The result has the value obtained by complementing \( I \) bit-by-bit according to the following truth table:

<table>
<thead>
<tr>
<th>( I )</th>
<th>NOT (( I ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The model for the interpretation of an integer value as a sequence of bits is in the section “The Bit Model”.

**Example**

If \( I \) is an integer of kind SHORT and has a value that is equal to 01010101 (base 2), NOT (\( I \)) has a value that is equal to 10101010 (base 2).

---

**OR(\( I \), \( J \))**

**Description**

Bitwise logical OR.

**Class**

Elemental nonstandard function.

**Arguments.**

- \( I \) must be of type integer.
- \( J \) must be of type integer with the same kind type parameter as \( I \).
Result Type and Type Parameter
Same as I.

Result Value
The result has the value obtained by performing an OR on I and J bit-by-bit according to the following truth table:

<table>
<thead>
<tr>
<th>I</th>
<th>J</th>
<th>OR(I, J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The model for interpreting an integer value as a sequence of bits is in the section “The Bit Model”.

Example
\( \text{OR}(3, 5) \) is 7. (Binary 0011 OR with binary 0101 is binary 0111.)

PACK(ARRAY, MASK, VECTOR)

Optional Argument
VECTOR

Description
Pack an array into an array of rank one under the control of a mask.

Class
Transformational function.
Arguments

ARRAY may be of any type. It must not be scalar.

MASK must be of type logical and must be conformable with ARRAY.

VECTOR (optional) must be of the same type and type parameters as ARRAY and must have rank one. VECTOR must have at least as many elements as there are .TRUE. elements in MASK. If MASK is scalar with the value .TRUE., VECTOR must have at least as many elements as there are in ARRAY.

Result Type, Type Parameter, and Shape

The result is an array of rank one with the same type and type parameters as ARRAY. If VECTOR is present, the result size is that of VECTOR; otherwise, the result size is the number t of .TRUE. elements in MASK unless MASK is scalar with the value .TRUE., in which case the result size is the size of ARRAY.

Result Value

Element i of the result is the element of ARRAY that corresponds to the ith .TRUE. element of MASK, taking elements in array element order, for i = 1, 2, ..., t. If VECTOR is present and has size n > t, element i of the result has the value VECTOR (i), for i = t+1, ..., n.

Examples

The nonzero elements of an array M with the value

\[
\begin{bmatrix}
  0 & 0 & 0 \\
  9 & 0 & 0 \\
  0 & 0 & ?
\end{bmatrix}
\]

may be “gathered” by the function PACK.

The result of PACK (M, MASK = M .NE. 0) is [9, 7].

The result of PACK (M, M .NE. 0, VECTOR = (/ 2, 4, 6, 8, 10, 12 /)) is [9, 7, 6, 8, 10, 12].
**PRECISION(X)**

**Description**
Returns the decimal precision in the model representing real numbers with the same kind type parameter as the argument.

**Class**
Inquiry function.

**Argument**
X must be of type real or complex. It may be scalar or array valued.

**Result Type, Type Parameter, and Shape**
Default integer scalar.

**Result Value**
The result has the value \( \text{INT}((p-1) \times \log_{10}(b)) + k \). The values of \( b \) and \( p \) are as defined in the section “The Real Number System Model” for the model representing real numbers with the same kind type parameter as X. The value of \( k \) is 1 if \( b \) is an integral power of 10 and 0 otherwise.

**Example.** PRECISION(X) has the value \( \text{INT}(23 \times \log_{10}(2.)) = \text{INT}(6.92...) = 6 \) for real X whose model is described in the section “The Real Number System Model”.

PRESENT(A)

**Description**
Determine whether an optional argument is present.

**Class**
Inquiry function.

**Argument**
A must be the name of an optional dummy argument that is accessible in the procedure in which the `PRESENT` function reference appears.

**Result Type and Type Parameters**
Default logical scalar.

**Result Value**
The result has the value `.TRUE.` if A is present and otherwise has the value `.FALSE.`.

**Example**
```
SUBROUTINE SUB (A, B, EXTRA)
    REAL A, B, C
    REAL, OPTIONAL :: EXTRA
    . . .
    IF (PRESENT (EXTRA)) THEN
        C = EXTRA
    ELSE
        C = (A+B)/2
    END IF
    . . .
END
```
If \texttt{SUB} is called with the statement
\begin{verbatim}
CALL SUB (10.0, 20.0, 30.0)
\end{verbatim}
then \(C\) is set to 30.0. If \texttt{SUB} is called with the statement
\begin{verbatim}
CALL SUB (10.0, 20.0)
\end{verbatim}
then \(C\) is set to 15.0.

An optional argument that is not present must not be referenced or defined
or supplied as a nonoptional actual argument, except as the argument of the
\texttt{PRESENT} intrinsic function.

\section*{PRODUCT(ARRAY, DIM, MASK)}

\subsection*{Optional Arguments}
\begin{description}
\item[DIM, MASK]
\end{description}

\subsection*{Description}
Product of all the elements of \texttt{ARRAY} along dimension \texttt{DIM} corresponding to
the \texttt{.TRUE.} elements of \texttt{MASK}.

\subsection*{Class}
Transformational function.

\subsection*{Arguments}
\begin{description}
\item[\texttt{ARRAY}]
must be of type integer, real, or complex. It must not be scalar.
\item[\texttt{DIM} (optional)]
must be scalar and of type integer with a value in the
range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of \texttt{ARRAY}. The
 corresponding actual argument must not be an optional dummy argument.
MASK (optional) must be of type logical and must be conformable with ARRAY.

**Result Type, Type Parameter, and Shape**

The result is of the same type and kind type parameter as ARRAY. It is scalar if DIM is absent or ARRAY has rank one; otherwise, the result is an array of rank n-1 and of shape \((d_1, d_2, \ldots, d_{DIM-1}, d_{DIM+1}, \ldots, d_n)\) where \((d_1, d_2, \ldots, d_n)\) is the shape of ARRAY.

**Result Value**

Case 1  The result of \(\text{PRODUCT}(\text{ARRAY})\) has a value equal to a processor-dependent approximation to the product of all the elements of ARRAY or has the value one if ARRAY has size zero.

Case 2  The result of \(\text{PRODUCT}(\text{ARRAY}, \text{MASK} = \text{msk})\) has a value equal to a processor-dependent approximation to the product of the elements of ARRAY corresponding to the .TRUE. elements of msk or has the value one if there are no .TRUE. elements.

Case 3  If ARRAY has rank one, \(\text{PRODUCT}(\text{ARRAY}, \text{DIM} [,\text{msk}])\) has a value equal to that of \(\text{PRODUCT}(\text{ARRAY} [,\text{MASK} = \text{ask}])\). Otherwise, the value of element \((s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_n)\) of \(\text{PRODUCT}(\text{ARRAY}, \text{DIM} [,\text{msk}])\) is equal to the following:

\[
\text{PRODUCT}(\text{ARRAY}(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_n) [,\text{MASK} = \text{msk}(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_n)])
\]

**Examples**

Case 1  The value of \(\text{PRODUCT}((/ 1, 2, 3 /))\) and \(\text{PRODUCT}((/ 1, 2, 3 /), \text{DIM} = 1)\) is 6.

Case 2  \(\text{PRODUCT}(C, \text{MASK} = C .\text{GT.} 0.0)\) forms the product of the positive elements of C.
Case 3 If \( B \) is the array

\[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix}
\]

then \( \text{PRODUCT}(B, \ DIM = 1) \) is \([2, 12, 30]\) and \( \text{PRODUCT}(B, \ DIM = 2) \) is \([15, 48]\).

---

**RADIX(X)**

**Description**
Returns the base of the model representing numbers of the same type and kind type parameter as the argument.

**Class**
Inquiry function.

**Argument**
\( X \) must be of type integer or real. It may be scalar or array valued.

**Result Type, Type Parameter, and Shape**
Default integer scalar.

**Result Value**
The result has the value \( r \) if \( X \) is of type integer and the value \( b \) if \( X \) is of type real, where \( r \) and \( b \) are as defined in the section “The Real Number System Model”.

**Example**
\( \text{RADIX}(X) \) has the value 2 for real \( X \) whose model is described in the section “The Real Number System Model”.
RANDOM_NUMBER(HARVEST)

Description
Returns one pseudorandom number or an array of pseudorandom numbers from the uniform distribution over the range $0 \leq x < 1$.

Class
Subroutine.

Argument
HARVEST must be of type real. It is an INTENT(OUT) argument. It may be a scalar or an array variable. It is set to contain pseudorandom numbers from the uniform distribution in the interval $0 \leq x < 1$.

Examples
REAL X, Y (10, 10)
! Initialize X with a pseudorandom number
CALL RANDOM_NUMBER (HARVEST = X)
CALL RANDOM_NUMBER (Y)
! X & Y contain
! uniformly distributed random numbers
RANDOM_SEED(SIZE, PUT, GET)

Optional Arguments
SIZE, PUT, GET

Description
Restarts or queries the pseudorandom number generator used by RANDOM_NUMBER.

Class
Subroutine.

Arguments
There must either be exactly one or no arguments present.

SIZE (optional) must be scalar and of type default integer. It is an INTENT (OUT) argument. It is set to the number \( N \) of integers that the processor uses to hold the value of the seed.

PUT (optional) must be a default integer array of rank one and size \( \geq N \). It is an INTENT (IN) argument. It is used by the processor to set the seed value.

GET (optional) must be a default integer array of rank one and size \( \geq N \). It is an INTENT (OUT) argument. It is set by the processor to the current value of the seed. If no argument is present, the processor sets the seed to a processor-dependent value.

Examples
CALL RANDOM_SEED
! Processor initialization
CALL RANDOM_SEED (SIZE = K) ! Sets K = N
CALL RANDOM_SEED (PUT = SEED (1 : K)) ! Set user seed
CALL RANDOM_SEED (GET = OLD (1 : K)) ! Read current seed
RANGE(X)

Description
Returns the decimal exponent range in the model representing integer or real numbers with the same kind type parameter as the argument.

Class
Inquiry function.

Argument
X must be of type integer, real, or complex. It may be scalar or array valued.

Result Type, Type Parameter, and Shape
Default integer scalar.

Result Value
Case 1 For an integer argument, the result has the value INT (LOG10 (huge)), where huge is the largest positive integer in the model representing integer numbers with same kind type parameter as X (see the section “The Integer Number System Model”).

Case 2 For a real or complex argument, the result has the value INT (MIN (LOG10 (huge), -LOG10 (tiny))), where huge and tiny are the largest and smallest positive numbers in the model representing real numbers with the same value for the kind type parameter as X (see the section “The Real Number System Model”).
Example
RANGE \(X\) has the value 38 for real \(X\), whose model is described in “The Real Number System Model”, because in this case \(huge = (1 - 2^{-24}) \times 2^{127}\) and \(tiny = 2^{-127}\).

**REAL(A, KIND)**

<table>
<thead>
<tr>
<th>Optional Argument</th>
<th>KIND</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Convert to real type.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Elemental function.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
<th>A must be of type integer, real, or complex.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KIND (optional) must be a scalar integer initialization expression.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result Type and Type Parameter</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Real</th>
</tr>
</thead>
</table>

Case 1 If A is of type integer or real and KIND is present, the kind type parameter is that specified by KIND. If A is of type integer or real and KIND is not present, the kind type parameter is the processor-dependent kind type parameter for the default real type.

Case 2 If A is of type complex and KIND is present, the kind type parameter is that specified by KIND.
If \( A \) is of type complex and \( \text{KIND} \) is not present, the kind type parameter is the kind type parameter of \( A \).

**Result Value**

Case 1  If \( A \) is of type integer or real, the result is equal to a processor-dependent approximation to \( A \).

Case 2  If \( A \) is of type complex, the result is equal to a processor-dependent approximation to the real part of \( A \).

**Examples**

\[
\text{REAL}(-3) \text{ has the value } -3.0.
\]

\[
\text{REAL}(Z) \text{ has the same kind type parameter and the same value as the real part of the complex variable } Z.
\]

\[
\text{REAL}(2.0\_\text{HIGH}/3.0) \text{ is } 0.66666666666666 \text{ with kind \text{HIGH}.}
\]

---

**REPEAT(STRING, NCOPIES)**

**Description**

Concatenate several copies of a string.

**Class**

Transformational function.

**Arguments**

- **STRING**  must be scalar and of type character.
- **NCOPIES**  must be scalar and of type integer. Its value must not be negative.
Result Type, Type Parameter, and Shape

Character scalar of length \( N \text{COPIES} \) times that of STRING, with the same kind type parameter as STRING.

Result Value

The value of the result is the concatenation of \( N \text{COPIES} \) copies of STRING.

Examples

\[
\text{REPEAT('H', 2)} \quad \text{has the value HH.}
\]
\[
\text{REPEAT('XYZ', 0)} \quad \text{has the value of a zero-length string.}
\]

RESHAPE(SOURCE, SHAPE, PAD, ORDER)

Optional Arguments

PAD, ORDER

Description

Constructs an array of a specified shape from the elements of a given array.

Class

Transformational function.

Arguments

SOURCE may be of any type. It must be array valued. If PAD is absent or of size zero, the size of SOURCE must be greater than or equal to \( \text{PRODUCT(SHAPE)} \). The size of the result is the product of the values of the elements of SHAPE.
SHAPE must be of type integer, rank one, and constant size. Its size must be positive and less than 8. It must not have an element whose value is negative.

PAD (optional) must be of the same type and type parameters as SOURCE. PAD must be array valued.

ORDER (optional) must be of type integer, must have the same shape as SHAPE, and its value must be a permutation of \([1, 2, \ldots, n]\), where \(n\) is the size of SHAPE. If absent, it is as if it were present with value \([1, 2, \ldots, n]\).

**Result Type, Type Parameter, and Shape**

The result is an array of shape SHAPE (that is, SHAPE(RESHAPE(SOURCE, SHAPE, PAD, ORDER)) is equal to SHAPE) with the same type and type parameters as SOURCE.

**Result Value**

The elements of the result, taken in permuted subscript order ORDER(1), \ldots, ORDER(n), are those of SOURCE in normal array element order followed if necessary by those of PAD in array element order, followed if necessary by additional copies of PAD in array element order.

**Examples**

RESHAPE((/ 1, 2, 3, 4, 5, 6 /), (/ 2, 3 /)) has the value

\[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6 \\
\end{bmatrix}
\]

RESHAPE((/ 1, 2, 3, 4, 5, 6 /), (/ 2, 4 /), (/ 0, 0 /), (/ 2, 1 /)) has the value

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 0 & 0 \\
\end{bmatrix}
\]
**RNUM(I)**

**Description**
Convert character to real type.

**Class**
Elemental nonstandard function.

**Argument**
I must be of type character.

**Result**
Default real type.

**Example**
\[ \text{RNUM("821.003") is 821.003 of type default real.} \]

**RRSPACING(X)**

**Description**
Returns the reciprocal of the relative spacing of model numbers near the argument value.

**Class**
Elemental function.
Argument
X must be of type real.

Result Type and Type Parameter
Same as X.

Result Value
The result has the value $|x \times b^{-e}| \times b^p$, where $b$, $e$, and $p$ are as defined in the section “The Real Number System Model”.

Example
RRSPACING(-3.0) has the value $0.75 \times 2^{24}$ for reals whose model is described in the section “The Real Number System Model”.

RSHFT(I, SHIFT)

Description
Bitwise right shift.

Class
Elemental nonstandard function.

For details see “RSHFT” on page 122.
**RSHIFT(I, SHIFT)**

**Description**
Bitwise right shift.

**Class**
Elemental nonstandard function.
For details see “RSHIFT” on page 123.

**SCALE(X, I)**

**Description**
Returns $x \times b^i$ where $b$ is the base in the model representation of $x$ (see the section “The Real Number System Model”).

**Class**
Elemental function.

**Arguments**
- $x$ must be of type real.
- $i$ must be of type integer.

**Result Type and Type Parameter**
Same as $x$. 
**Result Value**

The result has the value $x \times b^i$, where $b$ is defined in the section “The Real Number System Model”, provided this result is within range; if not, the result is processor dependent.

**Example**

`SCALE (3.0, 2)` has the value 12.0 for reals whose model is described in “The Real Number System Model”.

---

**SCAN(STRING, SET, BACK)**

---

**Optional Argument**

BACK

**Description**

Scan a string for any one of the characters in a set of characters.

**Class**

Elemental function.

**Arguments**

- STRING
  - must be of type character.
- SET
  - must be of type character with the same kind type parameter as STRING.
- BACK (optional)
  - must be of type logical.

**Result Type and Type Parameter**

Default integer.
**Result Value**

Case 1  
If `BACK` is absent or is present with the value `.FALSE.` and if `STRING` contains at least one character that is in `SET`, the value of the result is the position of the leftmost character of `STRING` that is in `SET`.

Case 2  
If `BACK` is present with the value `.TRUE.` and if `STRING` contains at least one character that is in `SET`, the value of the result is the position of the rightmost character of `STRING` that is in `SET`.

Case 3  
The value of the result is zero if no character of `STRING` is in `SET` or if the length of `STRING` or `SET` is zero.

**Examples**

Case 1  
`SCAN('FORTRAN', 'TR')` has the value 3.

Case 2  
`SCAN('FORTRAN', 'TR', BACK = .TRUE.)` has the value 5.

Case 3  
`SCAN('FORTRAN', 'BCD')` has the value 0.

---

**SELECTED_INT_KIND(R)**

**Description**

Returns a value of the kind type parameter of an integer data type that represents all integer values \( n \) with \( -10^R < n < 10^R \).

**Class**

Transformational function.

**Argument**

\( R \) must be scalar and of type integer.
Result Type, Type Parameter, and Shape
Default integer scalar.

Result Value
The result has a value equal to the value of the kind type parameter of an integer data type that represents all values \( n \) in the range of values \( n \) with \(-10^5 < n < 10^5\), or if no such kind type parameter is available on the processor, the result is \(-1\).
If more than one kind type parameter meets the criteria, the value returned is the one with the smallest decimal exponent range, unless there are several such values, in which case the smallest of these kind values is returned.

Example
SELECTED_INT_KIND(6) has the value KIND(0) on a machine that supports a default integer representation method with \( r=2 \) and \( q=31 \) as defined in the model for the integer number systems in “The Integer Number System Model”.

SELECTED_REAL_KIND(P, R)

Optional Arguments
P, R

Description
Returns a value of the kind type parameter of a real data type with decimal precision of at least \( P \) digits and a decimal exponent range of at least \( R \).

Class
Transformational function.
Arguments
At least one argument must be present.

\( P \) (optional) must be scalar and of type integer.

\( R \) (optional) must be scalar and of type integer.

Result Type, Type Parameter, and Shape
Default integer scalar.

Result Value
The result has a value equal to a value of the kind type parameter of a real data type with decimal precision, as returned by the function \texttt{PRECISION}, of at least \( P \) digits and a decimal exponent range, as returned by the function \texttt{RANGE}, of at least \( R \).

If no such kind type parameter is available on the processor, the result is \(-1\) if the precision is not available, \(-2\) if the exponent range is not available, and \(-3\) if neither is available.

If more than one kind type parameter value meets the criteria, the value returned is the one with the smallest decimal precision, unless there are several such values, in which case the smallest of these kind values is returned.

Example
\texttt{SELECTED_REAL_KIND(6, 70)} has the value \texttt{KIND(0.0)} on a machine that supports a default real approximation method with \( p=16, \ p=6, \ e_{\text{min}}=-64, \) and \( e_{\text{max}}=63 \) as defined in the model for the real number system in the section “The Real Number System Model”.
SET_EXPONENT(X, I)

Description
Returns the model number whose fractional part is the fractional part of the model representation of \( X \) and whose exponent part is \( I \).

Class
Elemental function.

Arguments
\( X \) must be of type real.
\( I \) must be of type integer.

Result Type and Type Parameter
Same as \( X \).

Result Value
The result has the value \( X \times b^{I-e} \), where \( b \) and \( e \) are as defined in the section “The Real Number System Model”, provided this result is within range; if not, the result is processor-dependent.
If \( X \) has value zero, the result has value zero.

Example
\( \text{SET\_EXPONENT}(3.0, 1) \) has the value 1.5 for reals whose model is as described in the section “The Real Number System Model”.
SHAPE(SOURCE)

**Description**
Returns the shape of an array or a scalar.

**Class**
Inquiry function.

**Argument**
SOURCE may be of any type. It may be array valued or scalar. It must not be a pointer that is disassociated or an allocatable array that is not allocated. It must not be an assumed-size array.

**Result Type, Type Parameter, and Shape**
The result is a default integer array of rank one whose size is equal to the rank of SOURCE.

**Result Value**
The value of the result is the shape of SOURCE.

**Examples**
The value of SHAPE(A(2:5, -1:1) ) is [4, 3].
The value of SHAPE(3) is the rank-one array of size zero.
SIGN(A, B)

**Description**
Absolute value of A times the sign of B.

**Class**
Elemental function.

**Arguments**
- A must be of type integer or real.
- B must be of the same type and kind type parameter as A.

**Result Type and Type Parameter**
Same as A.

**Result Value**
The value of the result is |A| if B ≥ 0 and −|A| if B < 0.

**Example**
SIGN(−3.0, 2.0) has the value 3.0.

SIN(X)

**Description**
Sine function in radians.
Class
Elemental function.

Argument
\( x \) must be of type real or complex.

Result Type and Type Parameter
Same as \( x \).

Result Value
The result has a value equal to a processor-dependent approximation to \( \sin(x) \).
If \( x \) is of type real, it is regarded as a value in radians.
If \( x \) is of type complex, its real part is regarded as a value in radians.

Examples
\( \sin(1.0) \) has the value 0.84147098.
\( \sin((0.5_{\text{HIGH}}, 0.5)) \) has the value 0.54061268571316 + 0.45730415318425i with kind \text{HIGH}.

SIND(X)

Description
Sine function that accepts input in degrees.

Class
Elemental nonstandard function.
**Argument**

X must be of type real.

**Result Type and Type Parameter**

Same as X.

**Result Value**

The result has a value equal to a processor-dependent approximation to \( \sin(X) \).

**Examples**

\( \text{SIND}(0.0) \) has the value 0.0.
\( \text{SIND}(30.0) \) has the value 0.5.

---

**SINH(X)**

**Description**

Hyperbolic sine function.

**Class**

Elemental function.

**Argument**

X must be of type real.

**Result Type and Type Parameter**

Same as X.
**Result Value**
The result has a value equal to a processor-dependent approximation to \( \sinh(x) \).

**Examples**
\[ \sinh(1.0) \] has the value 1.1752012.
\[ \sinh(0.5_{\text{HIGH}}) \] has the value 0.52109530549375 with kind \text{HIGH}.

---

**SIZE(ARRAY, DIM)**

**Optional Argument**

**DIM**

**Description**
Returns the extent of an array along a specified dimension or the total number of elements in the array.

**Class**
Inquiry function.

**Arguments.**

**ARRAY** may be of any type. It must not be scalar. It must not be a pointer that is disassociated or an allocatable array that is not allocated. If **ARRAY** is an assumed-size array, **DIM** must be present with a value less than the rank of **ARRAY**.

**DIM** (optional) must be scalar and of type integer with a value in the range \( 1 \leq \text{DIM} \leq n \), where \( n \) is the rank of **ARRAY**.

**Result Type, Type Parameter, and Shape**
Default integer scalar.
Result Value
The result has a value equal to the extent of dimension DIM of ARRAY or, if DIM is absent, the total number of elements of ARRAY.

Examples
The value of SIZE(A(2:5, -1:1), DIM=2) is 3.
The value of SIZE(A(2:5, -1:1) ) is 12.

SPACING(X)

Description
Returns the absolute spacing of model numbers near the argument value.

Class
Elemental function.

Argument
X must be of type real.

Result Type and Type Parameter
Same as X.

Result Value
If X is not zero, the result has the value $b^e \cdot p$, where $b$, $e$, and $p$ are as defined in the section “The Real Number System Model”, provided this result is within range; otherwise, the result is the same as that of TINY(X).

Example
SPACING(3.0) has the value $2^{-22}$ for reals whose model is described in the section “The Real Number System Model”.
Intrinsic Procedures

SPREAD(SOURCE, DIM, NCOPIES)

Description
Replicates an array by adding a dimension. Broadcasts several copies of SOURCE along a specified dimension (as in forming a book from copies of a single page) and thus forms an array of rank one greater.

Class
Transformational function.

Arguments
SOURCE may be of any type. It may be scalar or array valued. The rank of SOURCE must be less than 7.
DIM must be scalar and of type integer with value in the range 1 ≤ DIM ≤ n + 1, where n is the rank of SOURCE.
NCOPIES must be scalar and of type integer.

Result Type, Type Parameter, and Shape
The result is an array of the same type and type parameters as SOURCE and of rank n + 1, where n is the rank of SOURCE.
Case 1 If SOURCE is scalar, the shape of the result is (MAX (NCOPIES, 0)).
Case 2 If SOURCE is array valued with shape (d₁, d₂, ..., dₙ), the shape of the result is (d₁, d₂, ..., d₉DIM-1, MAX (NCOPIES, 0), dDIM, ..., dₙ).
Result Value

Case 1  If SOURCE is scalar, each element of the result has a value equal to SOURCE.

Case 2  If SOURCE is array valued, the element of the result with subscripts \((r_1, r_2, \ldots, r_{n+1})\) has the value SOURCE\((r_1, r_2, \ldots, r_{\text{DIM}-1}, r_{\text{DIM}+1}, \ldots, r_{n+1})\).

Examples

Case 1  SPREAD\("A", 1, 3\) is the character array \((/'A', 'A', 'A' '/')\).

Case 2  If A is the array \([2, 3, 4]\), SPREAD\((A, \text{DIM}=1, \text{NCOPIES} = \text{NC})\) is the array

\[
\begin{bmatrix}
2 & 3 & 4 \\
2 & 3 & 4 \\
2 & 3 & 4 \\
\end{bmatrix}
\]

if \text{NC} has the value 3 and is a zero-sized array if \text{NC} has the value 0.

\[
\text{SQRT}(X)
\]

Description

Square root.

Class

Elemental function.
**Argument**

X must be of type real or complex. If X is real, its value must be greater than or equal to zero.

**Result Type and Type Parameter**

Same as X.

**Result Value**

The result has a value equal to a processor-dependent approximation to the square root of X.

A result of type complex is the principal value with the real part greater than or equal to zero. When the real part of the result is zero, the imaginary part is greater than or equal to zero.

**Examples**

\[ \text{SQRT}(4.0) \text{ has the value 2.0.} \]

\[ \text{SQRT}(5.0\_	ext{HIGH}) \text{ has the value 2.23606774998 with kind HIGH.} \]

---

**SUM(ARRAY, DIM, MASK)**

---

**Optional Arguments**

DIM, MASK

**Description**

Sum all the elements of ARRAY along dimension DIM corresponding to the .TRUE. elements of MASK.

**Class**

Transformational function.
Arguments

**ARRAY** must be of type integer, real, or complex. It must not be scalar.

**DIM (optional)** must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of **ARRAY**. The corresponding actual argument must not be an optional dummy argument.

**MASK (optional)** must be of type logical and must be conformable with **ARRAY**.

Result Type, Type Parameter, and Shape

The result is of the same type and kind type parameter as **ARRAY**. It is scalar if **DIM** is absent of **ARRAY** has rank one; otherwise, the result is an array of rank \(n-1\) and of shape \((d_1, d_2, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)\) where \((d_1, d_2, \ldots, d_n)\) is the shape of **ARRAY**.

Result Value

**Case 1** The result of \(\text{SUM(ARRAY)}\) has a value equal to a processor-dependent approximation to the sum of all the elements of **ARRAY** or has the value zero if **ARRAY** has size zero.

**Case 2** The result of \(\text{SUM(ARRAY, MASK = msk)}\) has a value equal to a processor-dependent approximation to the sum of the elements of **ARRAY** corresponding to the .TRUE. elements of **msk** or has the value zero if there are no .TRUE. elements.

**Case 3** If **ARRAY** has rank one, \(\text{SUM(ARRAY, DIM [,msk]})\) has a value equal to that of \(\text{SUM(ARRAY [,MASK = msk])}\). Otherwise, the value of element \((s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)\) of \(\text{SUM(ARRAY, DIM [,msk])}\) is equal to the following:

\[
\text{SUM(ARRAY}(s_1, s_2, \ldots, s_{\text{DIM}-1}, : , s_{\text{DIM}+1}, \ldots, s_n) [, \\
\text{MASK=msk}(s_1, s_2, \ldots, s_{\text{DIM}-1}, : , s_{\text{DIM}+1}, \ldots, s_n)])
\]
**Examples**

Case 1  The value of \( \text{SUM}(\{1, 2, 3\}) \) and \( \text{SUM}(\{1, 2, 3\}, \text{DIM}=1) \) is 6.

Case 2  \( \text{SUM}(C, \text{MASK}=C \ .G.T. \ 0.0) \) forms the arithmetic sum of the positive elements of \( C \).

Case 3  If \( B \) is the array

\[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix}
\]

then \( \text{SUM}(B, \text{DIM}=1) \) is \([3, 7, 11]\) and \( \text{SUM}(B, \text{DIM}=2) \) is \([9, 12]\).

**SYSTEM_CLOCK(COUNT, COUNT_RATE, COUNT_MAX)**

**Optional Arguments**

COUNT, COUNT_RATE, COUNT_MAX

**Description**

Returns integer data from a real-time clock.

**Class**

Subroutine.

**Arguments**

COUNT (optional)  must be scalar and of type default integer. It is an \text{INTENT} (\text{OUT}) argument. It is set to a processor-dependent value based on the current
value of the processor clock or to −HUGE (0) if there is no clock. The processor-dependent value is incremented by one for each clock count until the value COUNT_MAX is reached and is reset to zero at the next count. It lies in the range 0 to COUNT_MAX if there is a clock.

COUNT_RATE (optional) must be scalar and of type default integer. It is an INTENT (OUT) argument. It is set to the number of processor clock counts per second, or to zero if there is no clock.

COUNT_MAX (optional) must be scalar and of type default integer. It is an INTENT (OUT) argument. It is set to the maximum value that COUNT can have, or to zero if there is no clock.

Example
If the processor clock is a 24-hour clock that registers time in 1-second intervals, at 11:30 A.M. the reference

CALL SYSTEM_CLOCK (COUNT = C, COUNT_RATE = R, COUNT_MAX = M)

sets C = 11 * 3600 + 30 * 60 = 41400, R = 1, and M = 24 * 3600 - 1 = 86399.

TAN(X)

Description
Tangent function, which accepts the input in radians.

Class
Elemental function.
**Argument**

x must be of type real.

**Result Type and Type Parameter**

Same as x.

**Result Value**

The result has a value equal to a processor-dependent approximation to tan(x), with x regarded as a value in radians.

**Examples**

TAN(1.0) has the value 1.5574077.

TAN(2.0_HIGH) has the value −2.1850398632615 with kind HIGH.

---

**TAND(x)**

**Description**

Tangent function that accepts the input in degrees.

**Class**

Elemental nonstandard function.

**Argument**

x must be of type real.

**Result Type and Type Parameter**

Same as x.
**Result Value**

The result has a value equal to a processor-dependent approximation to \( \tan(X) \).

**Examples**

\[
\begin{align*}
\text{TAND}(0.0) & \text{ has the value 0.0.} \\
\text{TAND}(45.0) & \text{ has the value 1.0.} \\
\text{TAND}(135.0) & \text{ has the value -1.0.}
\end{align*}
\]

---

**TANH(X)**

**Description**

Hyperbolic tangent function.

**Class**

Elemental function.

**Argument**

\( x \) must be of type real.

**Result Type and Type Parameter**

Same as \( x \).

**Result Value**

The result has a value equal to a processor-dependent approximation to \( \tanh(X) \).

**Examples**

\[
\begin{align*}
\text{TANH}(1.0) & \text{ has the value 0.76159416.}
\end{align*}
\]
TANH(2.0_HIGH) has the value 0.96402758007582 with kind HIGH.

**TINY(X)**

**Description**
Returns the smallest positive number in the model representing numbers of the same type and kind type parameter as the argument.

**Class**
Inquiry function.

**Argument**
$x$ must be of type real. It may be scalar or array valued.

**Result Type, Type Parameter, and Shape**
Scalar with the same type and kind type parameter as $x$.

**Result Value**
The result has the value

$$b^{e_{\min} - 1}$$

where $b$ and $e_{\min}$ are as defined in the section “The Real Number System Model”.

**Example**
TINY(X) has the value $2^{-127}$ for real $x$, whose model is described in the section “The Real Number System Model”.
TRANSFER(SOURCE, MOLD, SIZE)

Optional Argument
SIZE

Description
Returns a result with a physical representation identical to that of SOURCE but interpreted with the type and type parameters of MOLD.

Class
Transformational function.

Arguments
SOURCE may be of any type and may be scalar or array valued.
MOLD may be of any type and may be scalar or array valued.
SIZE (optional) must be scalar and of type integer. The corresponding actual argument must not be an optional dummy argument.

Result Type, Type Parameter, and Shape
The result is of the same type and type parameters as MOLD.

Case 1 If MOLD is a scalar and SIZE is absent, the result is a scalar.

Case 2 If MOLD is array valued and SIZE is absent, the result is array valued and of rank one. Its size is as small as possible such that its physical representation is not shorter than that of SOURCE.

Case 3 If SIZE is present, the result is array valued of rank one and size SIZE.
Result Value

If the physical representation of the result has the same length as that of \texttt{SOURCE}, the physical representation of the result is that of \texttt{SOURCE}.

If the physical representation of the result is longer than that of \texttt{SOURCE}, the physical representation of the leading part is that of \texttt{SOURCE} and the remainder is undefined.

If the physical representation of the result is shorter than that of \texttt{SOURCE}, the physical representation of the result is the leading part of \texttt{SOURCE}. If \(D\) and \(E\) are scalar variables such that the physical representation of \(D\) is as long as or longer than that of \(E\), the value of \texttt{TRANSFER(TRANSFER(E, D), E)} must be the value of \(E\).

If \(D\) is an array and \(E\) is an array of rank one, the value of \texttt{TRANSFER(TRANSFER(E, D), E, SIZE(E))} must be the value of \(E\).

Examples

Case 1 \texttt{TRANSFER(1082130432, 0.0)} has the value 4.0 on a processor that represents the values 4.0 and 1082130432 as the string of binary digits 0100 0000 1000 0000 0000 0000 0000 0000.

Case 2 \texttt{TRANSFER((/ 1.1, 2.2, 3.3 /), (/ 0.0, 0.0 /))} is a complex rank-one array of length two whose first element is (1.1, 2.2) and whose second element has a real part with the value 3.3. The imaginary part of the second element is undefined.

Case 3 \texttt{TRANSFER((/ 1.1, 2.2, 3.3 /), (/ 0.0, 0.0 /), 1)} has the value 1.1 + 2.2 \textit{i}, which is a rank-one array with one complex element.
TRANSPOSE(MATRIX)

Description
Transpose an array of rank two.

Class
Transformational function.

Argument
MATRIX may be of any type and must have rank two.

Result Type, Type Parameters, and Shape. The result is an array of the same type and type parameters as MATRIX and with rank two and shape \((n, m)\) where \((m, n)\) is the shape of MATRIX.

Result Value
Element \((i, j)\) of the result has the value \(\text{MATRIX}(j, i)\), \(i = 1, 2, ..., n; j = 1, 2, ..., m)\.

Example
If A is the array
\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]
then TRANSPOSE (A) has the value
\[
\begin{bmatrix}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9
\end{bmatrix}
\]
TRIM(STRING)

Description
Returns the argument with trailing blank characters removed.

Class
Transformational function.

Argument
STRING must be of type character and must be a scalar.

Result Type and Type Parameters
Character with the same kind type parameter value as STRING and with a length that is the length of STRING less the number of trailing blanks in STRING.

Result Value
The value of the result is the same as STRING except any trailing blanks are removed. If STRING contains no nonblank characters, the result has zero length.

Example
TRIM(‘bA bbA’ ) is ’bAbB’.
UBOUND(ARRAY, DIM)

Optional Argument
DIM

Description
Returns all the upper bounds of an array or a specified upper bound.

Class
Inquiry function.

Arguments
ARRAY may be of any type. It must not be scalar. It must not be a pointer that is disassociated or an allocatable array that is not allocated. If ARRAY is an assumed-size array, DIM must be present with a value less than the rank of ARRAY.
DIM (optional) must be scalar and of type integer with a value in the range 1 ≤ DIM ≤ n, where n is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.

Result Type, Type Parameter, and Shape.
The result is of type default integer. It is scalar if DIM is present; otherwise, the result is an array of rank one and size n, where n is the rank ofARRAY.
Result Value

Case 1 For an array section or for an array expression, other than a whole array or array structure component, `UBOUND (ARRAY, DIM)` has a value equal to the number of elements in the given dimension; otherwise, it has a value equal to the upper bound for subscript `DIM` of `ARRAY` if dimension `DIM` of `ARRAY` does not have size zero and has the value zero if dimension `DIM` has size zero.

Case 2 `UBOUND (ARRAY)` has a value whose \( i \)th component is equal to `UBOUND(ARRAY, i)`, for \( i = 1, 2, ..., n \), where \( n \) is the rank of `ARRAY`.

Examples

If the following statements are processed

```
REAL, TARGET :: A (2:3, 7:10)
REAL, POINTER, DIMENSION (::,) :: B, C, D
B => A; C => A(:, :)
ALLOCATE (D(-3:3, -7:7))
```

then

- `UBOUND (A)` is \([3, 10]\)
- `UBOUND (A, DIM = 2)` is 10
- `UBOUND (B)` is \([3, 10]\)
- `UBOUND (C)` is \([2, 4]\)
- `UBOUND (D)` is \([3, 7]\)
UNPACK(VECTOR, MASK, FIELD)

Description
Unpack an array of rank one into an array under the control of a mask.

Class
Transformational function.

Arguments
VECTOR may be of any type. It must have rank one. Its size must be at least $t$ where $t$ is the number of .TRUE. elements in MASK.
MASK must be array valued and of type logical.
FIELD must be of the same type and type parameters as VECTOR and must be conformable with MASK.

Result Type, Type Parameter, and Shape
The result is an array of the same type and type parameters as VECTOR and the same shape as MASK.

Result Value
The element of the result that corresponds to the $i$th .TRUE. element of MASK, in array element order, has the value VECTOR($i$) for $i=1, 2, ..., t$, where $t$ is the number of .TRUE. values in MASK. Each other element has a value equal to FIELD if FIELD is scalar or to the corresponding element of FIELD if it is an array.
Examples
Specific values may be “scattered” to specific positions in an array by using UNPACK. If M is the array
\[
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]
then V is the array [1, 2, 3],
and Q is the logical mask
\[
\begin{bmatrix}
  \cdot & T & \cdot \\
  T & \cdot & \cdot \\
  \cdot & \cdot & T
\end{bmatrix}
\]
where “T” represents .TRUE. and “.” represents .FALSE., then the result of \text{UNPACK}(V, \text{MASK} = Q, \text{FIELD} = M) has the value
\[
\begin{bmatrix}
  1 & 2 & 0 \\
  1 & 1 & 0 \\
  0 & 0 & 3
\end{bmatrix}
\]
and the result of \text{UNPACK}(V, \text{MASK} = Q, \text{FIELD} = 0) has the value
\[
\begin{bmatrix}
  0 & 2 & 0 \\
  1 & 0 & 0 \\
  0 & 0 & 3
\end{bmatrix}
\]
VERIFY(STRING, SET, BACK)

Optional Argument
BACK

Description
Verify that a set of characters contains all the characters in a string by identifying the position of the first character in a string of characters that does not appear in a given set of characters.

Class
Elemental function.

Arguments
STRING must be of type character.
SET must be of type character with the same kind type parameter as STRING.
BACK (optional) must be of type logical.

Result Type and Type Parameter
Default integer.
**Result Value**

Case 1  If `BACK` is absent or present with the value `.FALSE.` and if `STRING` contains at least one character that is not in `SET`, the value of the result is the position of the leftmost character of `STRING` that is not in `SET`.

Case 2  If `BACK` is present with the value `.TRUE.` and if `STRING` contains at least one character that is not in `SET`, the value of the result is the position of the rightmost character of `STRING` that is not in `SET`.

Case 3  The value of the result is zero if each character in `STRING` is in `SET` or if `STRING` has zero length.

**Examples**

Case 1  `VERIFY('ABBA', 'A')` has the value 2.

Case 2  `VERIFY('ABBA', 'A', BACK = .TRUE.)` has the value 3.

Case 3  `VERIFY('ABBA', 'AB')` has the value 0.

---

**XOR(I, J)**

**Description**

Bitwise exclusive OR.

**Class**

Elemental nonstandard function.
Arguments

I must be of type integer.
J must be of type integer with the same kind type parameter as I.

Result Type and Type Parameter

Same as I.

NOTE. See the section "IXOR(I, J)" for result value information and examples.
Portability Functions

This chapter describes the functions (in alphabetical order) that comprise the portability library (libPEPCF90.lib). These functions are made available to the compiler when you invoke the /4Yportlib (Windows*) or -Vaxlib (Linux*) option. The function prototypes are described using the INTERFACE block, which provide the required information to complete a call to the specified function. For descriptions of the INTERFACE block and the /4Yportlib (Windows) or -Vaxlib (Linux) option, see the Intel® Fortran Compiler User's Guide

When using these functions, take the following considerations in mind:

- These functions are not a built-in part of the Fortran language. If you have FUNCTION, SUBROUTINE, or COMMON block names in your program that conflict with a particular portability function that you wish to use, you may need to change the name of the global entity in your program, in order to access the portability function.

- These functions are in user name space, and are linked only when you use the /4Yportlib (Windows) or -Vaxlib (Linux) option, and only then when there is no global definition in your program that satisfies the global reference.

- You do not have to insert a USE statement to interface to these functions, as with some other vendor’s products. Instead, for certain routines you can specify an INTERFACE block in your program to describe the interface to the routine in the portability library that you are calling. Without an INTERFACE block, you may get incorrect results unless you are very careful with implicit typing.

- If you want to include interfaces for all the routines in your program, you can either:
include the file iflport.f90 from the INCLUDE directory of your distribution or
— add a USE IFLPORT statement to access the INTERFACES for all portability functions.

## ABORT

*Flushes and closes I/O buffers, and terminates program execution.*

### Prototype

```fortran
USE IFLPORT
or
INTERFACE
  SUBROUTINE ABORT (STRING)
    MS$ATTRIBUTES ALIAS:'abort_':::ABORT
    CHARACTER(LEN=*) , OPTIONAL , INTENT(IN) :: STRING
  END SUBROUTINE
END INTERFACE
```

**STRING**  
*Input; optional. CHARACTER(LEN=*). Specifies abort message at program termination.*

### Description

This subroutine aborts current process, closes all files, flushes and closes I/O buffers, and terminates program execution. When ABORT is called, the default message written to external unit 0 is "abort: Fortran Abort Called." This message can be followed by STRING which allows you to add a specific message.

### Example

```
!The following prints "abort: Fortran Abort Called"
CALL ABORT
```
The following prints "abort: Out of here!"
Call ABORT ("Out of here!")

ACCESS
Determines file access values

Prototype
USE IFLPORT
or
INTEGER(4) FUNCTION ACCESS(NAME, MODE)
CHARACTER(LEN=*) NAME, MODE
END FUNCTION ACCESS

NAME Input. CHARACTER(LEN=*). Name of the file whose accessibility is to be determined.
MODE Input. CHARACTER(LEN=*). Modes of accessibility to check for. MODE is a character string of length one or greater containing only the characters “r”, “w”, “x”, or “ ” (a blank). These characters are interpreted as follows:

Character Meaning
The characters within MODE can appear in any order.

Usage
result = ACCESS (filename, mode)

Results
The result is INTEGER(4), and is zero if all access permissions in MODE are true. If input values that you pass to the function are invalid, or if the file cannot be accessed in all of the modes specified, one of the following error codes is returned:
EACCES       Access denied;
EINVAL       The mode argument is invalid
ENOENT       File not found

These error values are derived from the possible values of ERRNO, in Microsoft* Visual C++*.
Symbolic values for these error codes are defined in iflport.f90. The filename argument can contain either forward or backward slashes for path separators.
On Windows* operating systems, all files are readable. A test for read permission always returns 0.

Example
OPEN (UNIT=1,FILE='IFLFILE.TXT',STATUS='UNKNOWN')
WRITE (1,10) 'THIS IS A TEST'
CLOSE(UNIT=1)
! checks for read and write permission on the file "IFLFILE.TXT"

J = ACCESS ('IFLFILE.TXT', 'rw')
PRINT *, J
! checks whether 'IFLFILE.TXT' is executable. It is not, since
! it does not end in .COM, .EXE, .BAT, or .CMD
J = ACCESS ('IFLFILE.TXT','x')
PRINT *, J
10 FORMAT(A)
END
ALARM

Causes a subroutine to begin execution after a specified amount of time has elapsed.

Prototype
USE IFLPORT

Usage
result = ALARM (TIME, PROC)

TIME Input. Integer. Specifies the time delay (in seconds) between the call to ALARM and the time when PROC is to begin execution. If TIME is 0, the alarm is turned off and no routine is called.

PROC Input. Name of the procedure to call. The procedure takes no argument and must be declared EXTERNAL.

Results
The return value is INTEGER(4). It is zero if no alarm is pending. An alarm is pending if it has already been set by a previous call to ALARM. If alarm is pending, the routine returns the number of seconds remaining until the previously set alarm is to go off, rounded up to the nearest second.

After ALARM is called and the timer starts, the calling program continues for TIME seconds. The calling program then suspends and calls PROC, which runs in another thread. When PROC finishes, the alarm thread terminates, the original thread resumes, and the calling program resets the alarm. Once the alarm goes off, it is disabled until set again.

If PROC performs I/O or otherwise uses the Fortran library, you need to compile it with one of the multithread libraries. For information on multithreading, see Intel® Fortran User’s Guide.

The thread that PROC runs in has a higher priority than any other thread in the process. All other threads are essentially suspended until PROC terminates, or is blocked on some other event, such as I/O.
No alarms can occur after the main process ends. If the main program finishes or any thread executes an `EXIT` call, then any pending alarm is deactivated before it has a chance to run.

**Example**

```fortran
USE IFLPORT
INTEGER(4) ISEC, ISTAT
EXTERNAL SUBPROG
ISEC = 4
WRITE *, 'SUBPROG will begin in ', ISEC, ' SECONDS'
ISTAT = ALARM (ISEC, SUBPROG)
```

**AMOD**

*Returns the remainder of division of the first argument by the second argument*

**Prototype**

```fortran
INTERFACE
   REAL(4) FUNCTION AMOD(A,P)! REAL MODULUS
     !MS$ATTRIBUTES ALIAS:'amod_':::AMOD
     REAL(4), INTENT(IN) :: A,P
   END FUNCTION AMOD
END INTERFACE
```

**Syntax**

- `result = AMOD (A, P)`
- **A**
  - Input. Must be of type `REAL(4)`. First argument.
- **P**
  - Input. Must have the same type and kind parameters as `A`. 
Results
The result type is REAL(4). If \( P \) is not equal to zero, the value of the result is:
\[
\text{result} = A - \text{INT}(A/P) \times P.
\]
If \( P \) is equal to zero, the result is undefined.

BEEPQQ
Invokes the speaker

Prototype
USE IFLPORT
or
INTERFACE
    SUBROUTINE BEEPQQ(FREQ,DUR)
        INTEGER(4) FREQ, DUR
    END SUBROUTINE
END INTERFACE

Description
Invokes the speaker at the specified frequency for the specified duration in milliseconds.

Usage
CALL BEEPQQ (FREQ, DUR)
FREQ INTEGER(4). Frequency of the tone
DUR Length of the tone in milliseconds.
BEEPQQ does not return until the sound terminates.

Example
USE IFLPORT
INTEGER(4) FREQ, DUR
FREQ = 4000
DUR  = 1000
CALL BEEPQQ(FREQ, DUR)
END

BESJ0, BESJ1, BESJN, BESY0, BESY1, BESYN

Compute the single-precision values of Bessel functions

Prototype
USE IFLPORT
or
INTERFACE
REAL(4) FUNCTION BESJ0(X)
REAL(4) X
END FUNCTION

REAL(4) FUNCTION BESJ1(X)
REAL(4) X
END FUNCTION

REAL(4) FUNCTION BESJN(N,X)
INTEGER(4) N
REAL(4) X
END FUNCTION

REAL(4) FUNCTION BESY0(X)
REAL(4) X
END FUNCTION
REAL(4) FUNCTION BESY1(X)
REAL(4) X
END FUNCTION

REAL(4) FUNCTION BESYN(N,X)
INTEGER(4) N
REAL(4) X
END FUNCTION
END INTERFACE

Description
Compute the single-precision values of Bessel functions of the first and second kinds.

Usage
result = BESJ0 (REALPOSITIVE)
result = BESJ1 (REALPOSITIVE)
result = BESJN (n, REALPOSITIVE)
result = BESY0 (REALPOSITIVE)
result = BESY1 (REALPOSITIVE)
result = BESYN (n, REALPOSITIVE)

REALPOSITIVE REAL(4). Independent variable for a Bessel function. Must be greater than or equal to zero.

N INTEGER(4) unless changed by the user. Specifies the order of the selected Bessel function computation.

Results
BESJ0, BESJ1, and BESJN return Bessel functions of the first kind, orders 0, 1, and n, respectively, with the independent variable REALPOSITIVE.

BESY0, BESY1, and BESYN return Bessel functions of the second kind, orders 0, 1, and n, respectively, with the independent variable REALPOSITIVE.

Negative arguments return QNAN.
BIC, BIS

Perform a bit-level set and clear for integers.

Prototype
USE IFLPORT

Syntax
CALL BIC (BITNUM, TARGET)

BITNUM Input. INTEGER(4). Bit number to set. Must be in the range of 0 (least significant bit) to 31 (most significant bit).

TARGET Input. INTEGER(4). Variable in which to set a bit.

Description
BIC sets bit BITNUM of TARGET to 0; BIS sets bit BITNUM of TARGET to 1.

BIT

Performs a bit-level test for integers.

Prototype
USE IFLPORT

Syntax
result = BIT (BITNUM, SOURCE)
### BITNUM
Input. INTEGER (4). Bit number to test. Must be in the range of 0 (least significant bit) to 31 (most significant bit).

### SOURCE
Input. INTEGER (4). Variable being tested.

#### Output
The result is of type LOGICAL. It is .TRUE. if bit BITNUM of SOURCE is 1; otherwise, .FALSE..

### BSEARCHQQ
*Performs a binary search of a sorted one-dimensional array for a specified element. The array elements cannot be structures.*

#### Prototype
\[ \text{USE IFLPORT} \]

#### Syntax
\[ \text{result} = \text{BSEARCHQQ (ADRKEY, ADRARR, LENGTH, SIZE)} \]

**ADRKEY**
Input. INTEGER (4) for IA-32 systems; INTEGER (8) for Itanium-based systems. Address of the variable containing the element to be found (returned by LOC(X)).

**ADRARR**
Input. INTEGER (4) for IA-32 systems; INTEGER (8) for Itanium®-based systems. Address of the array (returned by LOC(X)).

**LENGTH**
Input. INTEGER (4). Number of elements in the array.
SIZE Input. INTEGER(4). Positive constant less than 32,767 that specifies the kind of array to be sorted. The following table lists constants, which are defined in `INCLUDE\DFLIB.F90` and specify type and kind for numeric arrays.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type of Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRT$INTEGER1</td>
<td>INTEGER(1)</td>
</tr>
<tr>
<td>SRT$INTEGER2</td>
<td>INTEGER(2) or equivalent</td>
</tr>
<tr>
<td>SRT$INTEGER4</td>
<td>INTEGER(4) or equivalent</td>
</tr>
<tr>
<td>SRT$REAL4</td>
<td>REAL(4) or equivalent</td>
</tr>
<tr>
<td>SRT$REAL8</td>
<td>REAL(8) or equivalent</td>
</tr>
</tbody>
</table>

If the value provided in `SIZE` is not a symbolic constant, and is less than 32,767, the array is assumed to be a character array with `SIZE` characters per element.

Output INTEGER(4). Array index of the matched entry or 0 if the entry is not found. The array must be sorted in ascending order before being searched.

**CAUTION.** You must make certain that the `LENGTH` and `SIZE` arguments are correct and the `SIZE` is the same for the element to be found and the array searched. If you pass invalid arguments, `BSEARCHQQ` attempts to search random parts of memory. If the memory it attempts to search is not allocated to the current process, the program is halted, and you receive the General Protection Violation message.

Example

```fortran
USE DFLIB
INTEGER(4) ARRAY(10), LENGTH
INTEGER(4) RESULT, TARGET
LENGTH = SIZE (ARRAY)
```
CDFLOAT

Converts an COMPLEX(4) to a DOUBLE PRECISION type

Prototype

INTERFACE
  DOUBLE PRECISION (REAL(8)) FUNCTION CDFLOAT (INPUT)
  COMPLEX(4), INTENT(IN)::INPUT
END FUNCTION CDFLOAT
END INTERFACE

INPUT         a COMPLEX(KIND=4) value

Description

CDFLOAT is an elemental function that converts a COMPLEX(KIND=4) type to DOUBLE PRECISION(REAL(8)).

Output

The COMPLEX value converted to DOUBLE PRECISION.
CHANGEDIRQQ

Sets specified directory to the current directory

Prototype

USE IFLPORT

or

INTERFACE

   LOGICAL(4) FUNCTION CHANGEDIRQQ(DIR)
   CHARACTER(LEN=*) DIR

END FUNCTION

END INTERFACE

Description

Sets the specified directory to the current, default directory.

Usage

result = CHANGEDIRQQ (DIR)

DIR CHARACTER(LEN=*). Directory to be made the current directory.

Results

LOGICAL(4). .TRUE. if successful; otherwise, .FALSE..

If you do not specify a drive in the DIR string, the named directory on the current drive becomes the current directory. If you specify a drive in DIR, the named directory on the specified drive becomes the current directory.

Example

USE IFLPORT

LOGICAL(4) CHANGEDIT

CHANGEDIT = CHANGEDIRQQ('c:\users')
CHANGEDRIVEQQ

Sets default drive

Prototype

USE IFLPORT
or

INTERFACE
    LOGICAL(4) FUNCTION CHANGEDRIVEQQ(DriveName)
    CHARACTER (LEN=*) DriveName
END FUNCTION
END INTERFACE

Description

Sets the specified drive to the current, default drive.

Usage

result = CHANGEDRIVEQQ (DriveName)
DriveName CHARACTER(LEN=*) CHARACTER value beginning with the drive letter.

Results

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

Drives are identified by a single alphabetic character. CHANGEDRIVEQQ examines only the first character of DriveName. The drive letter can be uppercase or lowercase.
CHANGEDRIVEQQ changes only the current drive. The current directory on the specified drive becomes the new current directory. If no current directory has been established on that drive, the root directory of the specified drive becomes the new current directory.

On Linux platforms, this routine always returns `.false.`.

**Example**

```fortran
USE IFLPORT
LOGICAL(4) CHANGEDIT
CHANGEDIT = CHANGEDRIVEQQ('d')
IF (CHANGEDIT) THEN
    PRINT *, 'CHANGEDRIVEQQ SUCCESSFUL'
ELSE
    PRINT *, 'Drive could not be changed'
ENDIF
END
```

---

**CHDIR**

_Changes the default directory._

**Prototype**

```fortran
USE IFLPORT
or
INTERFACE
INTEGER(4) FUNCTION CHDIR(DIRECTORY_NAME)
    CHARACTER(LEN=*) DIRECTORY_NAME
END FUNCTION CHDIR
END INTERFACE
```

**Usage**

```fortran
result = CHDIR(NEWDIRECTORY)
```
NEW_DIRECTORY CHARACTER(LEN=*) . Name of directory to become the default directory.

**Results**
The result type is INTEGER(4). CHDIR returns zero if the directory was changed successfully; otherwise, an error code. Possible error codes are:

- ENOENT: The named directory does not exist.
- ENOTDIR: The NEW_DIRECTORY parameter is not a directory.

**Example**
USE IFLPORT
CHARACTER(LEN=16) NEW_DIRECTORY
LOGICAL(4) CHANGEDIT
NEW_DIRECTORY='c:\program files'
CHANGEDIT=CHDIR(NEW_DIRECTORY)
IF (CHANGEDIT) THEN
   PRINT *, 'CHDIR SUCCESSFUL'
ELSE
   PRINT *,'Directory could not be changed'
ENDIF
END

---

CHMOD
Changes the access mode of a file.

**Prototype**
USE IFLPORT

**Syntax**
result = CHMOD (NAME, MODE)
NAME  Input. CHARACTER (LEN=*) . Name of the file whose access mode is to be changed. Must have a single path.

MODE  Input. CHARACTER (LEN=*) . File permission: either READ, WRITE, or EXECUTE. The MODE parameter can be either symbolic or absolute. An absolute mode is specified with an octal number consisting of any combination of the permission bits listed in the following table.

<table>
<thead>
<tr>
<th>Permission Bit</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>Set user ID on execution</td>
<td>Ignored; never true</td>
</tr>
<tr>
<td>2000</td>
<td>Set group ID on execution</td>
<td>Ignored; never true</td>
</tr>
<tr>
<td>1000</td>
<td>Sticky bit</td>
<td>Ignored; never true</td>
</tr>
<tr>
<td>0400</td>
<td>Read by owner</td>
<td>Ignored; always true</td>
</tr>
<tr>
<td>0200</td>
<td>Write by owner</td>
<td>Settable</td>
</tr>
<tr>
<td>0100</td>
<td>Execute by owner</td>
<td>Ignored; based on filename extension</td>
</tr>
<tr>
<td>0040, 0020,</td>
<td>Read, Write, Execute by group</td>
<td>Ignored; assumes owner permissions</td>
</tr>
<tr>
<td>0010</td>
<td>0040, 0020, 0010</td>
<td></td>
</tr>
<tr>
<td>0004, 0002,</td>
<td>Read, Write, Execute by others</td>
<td>Ignored; assumes owner permissions</td>
</tr>
<tr>
<td>0001</td>
<td>0004, 0002, 0001</td>
<td></td>
</tr>
</tbody>
</table>

The following regular expression represents a symbolic mode:

```
[ugoa]* [+ – =] [rwxXst]*
```

' [ugoa]*' is ignored; ' [+ – =]' indicates the operation to carry out:

- ' + ' Add the permission
- ' – ' Remove the permission
- ' = ' Absolutely set the permission

' [rwxXst] ' indicates the permission to add, subtract, or set. Only ' w ' is significant and affects write permission. All other letters are ignored.
Output
The result is INTEGER(4). Zero if the mode was changed successfully; otherwise an error code, some of which are:

- **ENOENT**: The specified file was not found.
- **EINVAL**: The mode argument is invalid
- **EPERM**: Permission denied; the file’s mode cannot be changed.

Example
USE IFLPORT
INTEGER(4) I, ISTATUS
I = ACCESS ('DATAFILE.TXT', 'w')
IF I then
    ISTATUS = CHMOD ('DATAFILE.TXT', '[+w]')
ENDIF
I = ACCESS ('DATAFILE.TXT', 'w')
PRINT *, I

**CLEARSTATUSFPQQ**
*Clears floating point processor status word.*

Prototype

```
INTERFACE
    SUBROUTINE CLEARSTATUSFPQQ()
    END SUBROUTINE
END INTERFACE
```

Description
Clear floating point processor status word.
**CLOCK**

*Returns current time*

**Prototype**

```
INTERFACE
  CHARACTER(LEN=8) FUNCTION CLOCK()
END FUNCTION
END INTERFACE
```

**Description**

Returns the current time of the day in the form `HH:MM:SS` using a 24-hour clock.

**Output**

The current time in the form of `hh:mm:ss`.

---

**CLOCKX**

*Returns processor clock*

**Prototype**

```
INTERFACE
  SUBROUTINE CLOCKX (CLOCK)
    REAL(8) CLOCK
  END SUBROUTINE CLOCKX
END INTERFACE
```

CLOCK current time
**Description**
This function returns the processor clock to the nearest microsecond.

**Output**
Processor clock to the nearest microsecond.

**COMMITQQ**
*Forces execution of any pending write operation(s)*

**Prototype**
```
USE IFLPORT
Or
INTERFACE
LOGICAL(4) FUNCTION COMMITQQ(LUN)
   INTEGER(4) LUN
END FUNCTION COMMITQQ
END INTERFACE
```

**Description**
Forces the operating system to execute any pending write operation(s) for the file associated with a specified unit to the file's physical device. Same functionality as `FLUSH`.

**Usage**
```
result = COMMITQQ (LUN)
LUN    Input. INTEGER(4). Fortran logical unit (LUN) attached to a file to be flushed from cache memory to a physical device.
```
2

Intel Fortran Libraries Reference

Results
The result type is LOGICAL(4). If an open LUN number is supplied, .TRUE. is returned and uncommitted records (if any) are written. If an unopened LUN is supplied, .FALSE. is returned.

Data written to files is often written into buffers, and held until the buffer is full. Only when the buffer is full, is the data written to the device. Data in the buffer is automatically flushed to disk when the file is closed. However, if the program or the computer crashes before the data is transferred from buffers, the data can be lost.

COMMITQQ forces any cached data intended for a file on a physical device to be written to that device immediately. This is called flushing the file.

NOTE. Copy the file iflport.f90 from the distribution list into a local directory and compile it before USE.

Example
USE IFLPORT
INTEGER LUN / 10 /
INTEGER len
CHARACTER(80) stuff
OPEN(LUN, FILE='COMMITQQ.TST', ACCESS='Sequential')
DO WHILE (.TRUE.)
  WRITE (*, '(A,' ')') 'Enter some data (Hit RETURN to exit): ' 
  len = GETSTRQQ (stuff)
  IF (len .EQ. 0) EXIT
  WRITE (LUN, *) stuff
  IF (.NOT. COMMITQQ(LUN)) WRITE (*,*) 'Failed'
END DO
CLOSE (LUN)
END
COMPL

*Returns a BIT-WISE Complement or logical .NOT. of the input value*

**Prototype**

```
INTERFACE COMPL

   INTEGER(4) FUNCTION COMPLINT(INVAL)
      INTEGER(4), INTENT(IN) :: INVAL
   END FUNCTION

   REAL(4) FUNCTION COMPLREAL(INVAL)
      REAL(4), INTENT(IN) :: INVAL
   END FUNCTION

   LOGICAL(4) FUNCTION COMPLLOG(INVAL)
      LOGICAL(4), INTENT(IN) :: INVAL
   END FUNCTION

END INTERFACE
```

INVAL  Input. INTEGER(4), REAL(4) or LOGICAL(4) for each of the above functions, respectively.

**Description**

Performs a BIT-WISE complement for INTEGER or REAL input or logical .NOT. for logical input.

**Output**

If the input is logical, the result is logical. Otherwise, the result is Boolean - a CRAY* bitset. With a Boolean result, use a BIT-WISE complement. For the logical COMPL, just toggle 1<-->0.
CSMG
Performs a BIT-WISE store under mask.

Prototype

```
INTERFACE
  INTEGER(4) FUNCTION CSMG(X, Y, Z)
  !MS$ATTRIBUTES ALIAS:’csmg_:’::CSMG
  INTEGER(4), INTENT(IN) :: X, Y, Z
END FUNCTION
END INTERFACE
```

Input
X, Y, Z

Description
Performs effective BIT-WISE store under mask.

The function returns the value based on the following rule: when a bit in Z is 1, the output bit is taken from X. When a bit in Z is zero, the corresponding output bit is taken from Y.

Output

The result type is INTEGER(4). The result is equal to the expression (x & z) | (y & ~z) where “&” is a bitwise AND operation, | - bitwise OR, ~ - bitwise NOT.
CTIME

Converts a given time into a 24-character ASCII string

Prototype

IA-32 systems

INTERFACE
  CHARACTER(LEN=24) FUNCTION CTIME(TIME)
  INTEGER(4) INTENT(IN) :: TIME
END FUNCTION CTIME
END INTERFACE

Itanium®-based systems

INTERFACE
  CHARACTER(LEN=24) FUNCTION CTIME(TIME)
  INTEGER(8) INTENT(IN) :: TIME
END FUNCTION CTIME
END INTERFACE

TIME elapsed time in seconds since 00:00:00 Greenwich Mean Time, January 1, 1970.

Description

This function takes an INTEGER(4) for IA-32 or INTEGER(8) for Itanium-based systems variable or expression as input. The input value is some number of seconds since midnight of January 1, 1970, in Greenwich Mean Time. CTIME converts this integer input value into an ASCII character string.

Output

A 24-character ASCII string in the form: Thu Jan 15 00:00:01 1970.
DATE

Returns the current date and time as ASCII string

INTERFACE
  SUBROUTINE DATE(DATESTR)
  CHARACTER(LEN=9) DATESTR
  END SUBROUTINE DATE
END INTERFACE

Description

This function returns the current date and time as a nine character ASCII string.

Output

A 9-character ASCII string in the form: dd–mmm–yy where:

dd is the 2-digit date

mmm is the 3 letter month

yy is the last two digits of the year

WARNING. This routine may cause problems with the year 2000. Use DATE_AND_TIME or DATE4 instead.

Example

The following program gets and prints the current system date. Its output could be, for example, “12-Jul-96” (without the quotes).

PROGRAM datetest
  CHARACTER(9) :: today
  INTRINSIC DATE

  WRITE(*,*) DATE
CALL DATE(today)
PRINT *, today
END

**DATE4**

*Returns the current date and time as ASCII string*

**Prototype**

```fortran
INTERFACE
    SUBROUTINE DATE4(DATESTR)
       CHARACTER(LEN=11) DATESTR
    END SUBROUTINE DATE4
END INTERFACE
```

**Description**

This function returns the current date and time as an eleven character ASCII string.

**Output**

An 11-character ASCII string in the form: *dd-mm-yyyy* where:

- **dd**: is the 2-digit date
- **mmm**: is the 3 letter month
- **yy**: is the last two digits of the year

**NOTE.** *Although using DATE_AND_TIME is better, since it is now standard in the Fortran language, this routine gives you the functionality of the old DATE routine and is year-2000 compliant.*
DBESJ0, DBESJ1, DBESJN, DBESY0, DBESY1, DBESYN

Compute the double-precision numbers of Bessel functions.

Prototype

USE IFLPORT or INTERFACE

REAL(8) FUNCTION DBESJ0(X)
REAL(8) X
END FUNCTION

REAL(8) FUNCTION DBESJ1(X)
REAL(8) X
END FUNCTION

REAL(8) FUNCTION DBESJN(N,X)
INTEGER(4) N
REAL(8) X
END FUNCTION

REAL(8) FUNCTION DBESY0(X)
REAL(8) X
END FUNCTION

REAL(8) FUNCTION DBESY1(X)
REAL(8) X
END FUNCTION

REAL(8) FUNCTION DBESYN(N,X)
INTEGER(4) N
REAL(8) X
END FUNCTION
END INTERFACE

Descriptions
Compute the double-precision values of Bessel functions of the first and second kinds.

Usage
result = DBESJ0 (DOUBLEPOS)
result = DBESJ1 (DOUBLEPOS)
result = DBESJN (n, DOUBLEPOS)
result = DBESY0 (DOUBLEPOS)
result = DBESY1 (DOUBLEPOS)
result = DBESYN (n, DOUBLEPOS)

DOUBLEPOS REAL(8). Independent variable for a Bessel function. Must be greater than or equal to zero.

N Integer. Specifies the order of the selected Bessel function computation.

Results
DBESJ0, DBESJ1, and DBESJN return Bessel functions of the first kind, orders 0, 1, and \( n \), respectively, with the independent variable DOUBLEPOS.

DBESY0, DBESY1, and DBESYN return Bessel functions of the second kind, orders 0, 1, and \( n \), respectively, with the independent variable DOUBLEPOS.

Negative arguments cause DBESY0, DBESY1, and DBESYN to return a huge negative value.

Example
    USE IFLPORT
    REAL(8) BESNUM, BESOUT
    10 READ *, BESNUM
       BESOUT = DBESJ0(BESNUM)
       PRINT *, 'Result is ', BESOUT
    GOTO 10
DCLOCK

Provides elapsed time in seconds since the start of the current process.

Prototype

FUNCTION DCLOCK()
END FUNCTION

Description

This function returns the elapsed time since the start of your process as a DOUBLE PRECISION number.

Note: The first call to DCLOCK performs calibration.

Class
Elemental nonstandard function.

Result Type and Type Parameter

DOUBLE PRECISION

Output

The time in seconds since the beginning of your process. This routine provides accurate timing to the nearest microsecond, taking into account the frequency of the processor where the current process is running. You can obtain equivalent results using standard Fortran by using the CPU_TIME intrinsic function.

Examples

DOUBLE PRECISION START_TIME, STOP_TIME, DCLOCK
EXTERNAL DCLOCK
START_CLOCK = DCLOCK()
CALL FOO()
STOP_CLOCK = DCLOCK()
PRINT *, 'foo took: ', STOP_CLOCK - START_CLOCK, ' seconds.'

DELDIRQQ
Deletes a specified directory

Prototype
USE IFLPORT
or

INTERFACE
   LOGICAL(4) FUNCTION DELDIRQQ(DirName)
       CHARACTER(LEN=*) DirName
   END FUNCTION
END INTERFACE

Usage
result = DELDIRQQ (DIRNAME)

DIRNAME CHARACTER(LEN=*) CHARACTER value containing
the name of the directory to be deleted.

Results
The result is LOGICAL(4). The result is .TRUE. if successful; otherwise,
.FALSE..

The directory to be deleted must be empty. It cannot be the current
directory, the root directory, or a directory currently in use by another
process.
DELFFILESQQ

*Deletes files matching specification*

**Prototype**

```fortran
USE IFLPORT
or
INTERFACE
  INTEGER(4) FUNCTION DELFILESQQ(FILESPEC)
  CHARACTER(LEN=*) FILESPEC
END FUNCTION
END INTERFACE
```

**Description**

Deletes all files matching the name specification, which can contain wildcards (* and ?).

**Usage**

```fortran
result = DELFILESQQ (FILESPEC)
FILESPEC CHARACTER(LEN=*) . File(s) to be deleted. Can contain wildcards (* and ?).
```

**Results**

The result type is INTEGER(2). The return value is the number of files deleted.

You can use wildcards to delete more than one file at a time. DELFILESQQ does not delete directories or system, hidden, or read-only files. Use this function with caution because it can delete many files at once. If a file is in use by another process (for example, if it is open in another process), it cannot be deleted.

**Example**

```fortran
USE IFLPORT
```
INTEGER(4) len, count  
CHARACTER(80) file  
CHARACTER(1) ch  
WRITE(*,*) 'Enter names of files to delete: '  
len = GETSTRQQ(file)  
IF (file(1:len) .EQ. '*.*') THEN  
   WRITE(*,*) 'Are you sure (Y/N)?
   ch = GETCHARQQ()  
   IF ((ch .NE. 'Y') .AND. (ch .NE. 'y')) THEN  
      STOP 'No files deleted'
   ELSE
      PRINT *, 'OK, deleting all files'
   ENDIF  
ENDIF  
END IF  
count = DELFILESQQ(file)  
WRITE(*,*) 'Deleted ', count, ' files.'  
END

**DFLOATI**

*Converts an INTEGER(2) to a DOUBLE PRECISION type*

**Prototype**

```fortran
INTERFACE
   DOUBLE PRECISION (REAL(8)) FUNCTION DFLOATI (INPUT)
   INTEGER(2), INTENT(IN)::INPUT
END FUNCTION DFLOATI
END INTERFACE
```

**INPUT**

A scalar INTEGER(KIND=2) value
**Description**

DFLOATI is an elemental function that converts a scalar integer (KIND=2) type to DOUBLE PRECISION (REAL(8)).

**Output**

The integer value converted to DOUBLE PRECISION.

---

**DFLOATJ**

*Converts an INTEGER(4) to a DOUBLE PRECISION type*

**Prototype**

```fortran
INTERFACE
  DOUBLE PRECISION (REAL(8)) FUNCTION DFLOATJ (INPUT)
  INTEGER(4), INTENT(IN)::INPUT
END FUNCTION DFLOATJ
END INTERFACE
```

**Description**

DFLOATJ is an elemental function that converts an INTEGER(4) type to DOUBLE PRECISION (REAL(8)) type.

**Output**

The integer value converted to DOUBLE PRECISION.
DFLOATK

Converts an INTEGER(8) type to a DOUBLE PRECISION type

Prototype

INTERFACE
   REAL(8) FUNCTION DFLOATK (INPUT)
   INTEGER(8), INTENT(IN)::INPUT
END FUNCTION DFLOATK
END INTERFACE

INPUT
an INTEGER(8) value or expression.

Description

DFLOATK is an elemental function that converts an INTEGER(8) type to a DOUBLE PRECISION (or REAL(8)) type.

Output

The integer value converted to DOUBLE PRECISION.

DMOD

Returns the remainder of division of the first argument by the second argument

Prototype

INTERFACE
   REAL(8) FUNCTION DMOD(A,P)
   !MSSATTRIBUTES ALIAS:'dmod_'::DMOD
   REAL(8), INTENT(IN) :: A,P
END FUNCTION DMOD
END INTERFACE
END FUNCTION
END INTERFACE

Syntax

\texttt{result = DMOD (A, P)}

\textbf{A} \hspace{1cm} \text{Input. Must be of type REAL(8). First argument.}

\textbf{P} \hspace{1cm} \text{Input. REAL(8). Must have the same type and kind parameters as A.}

Results

The result type is REAL(8). If \( P \) is not equal to zero, the value of the result is:

\[ \text{result} = A - \text{INT}(A/P) \times P. \]

If \( P \) is equal to zero, the result is undefined.

---

\textbf{DRAND}

*Generates successive pseudorandom numbers in the range of 0. to 1.*

Prototype

\text{INTERFACE}

\texttt{REAL(8) FUNCTION DRAND(NEW_SEED)}

!MS$ATTRIBUTES ALIAS:'drand_':::DRAND
\texttt{INTEGER(4), INTENT(IN)::NEW_SEED}

\texttt{END FUNCTION DRAND}

\text{END INTERFACE}

Description

Generate successive pseudorandom numbers uniformly distributed in the range of 0.0 to 1.0.
Class
Elemental nonstandard function.

Result Type and Type Parameter
REAL(8) type.

Example
REAL(8) rv
rv = RAND()

NOTE. For details about restarting the pseudorandom number generator used by IRAND and RAND, see the SRAND section.

DRANDM
Generates DOUBLE-PRECISION random numbers in the range of 0. to 1.

Prototype
USE IFILPORT
or
INTERFACE
   REAL(8) FUNCTION DRANDM(NEW_SEED)
   !MSATTRIBUTES ALIAS:'drand_'::DRANDM
   INTEGER(4), INTENT(IN)::NEW_SEED
END FUNCTION DRANDM
END INTERFACE

NEW_SEED Input. INTEGER (4). Controls the way random number is selected.
**Syntax**

```fortran
result = DRANDM (NEW_SEED)
```

NEW_SEED  
Input. INTEGER(4). Controls the way random number is selected.

**Output**

The result is of type REAL(8). The values are:

<table>
<thead>
<tr>
<th>NEW_SEED value</th>
<th>Selection Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The generator is restarted and the first value is selected</td>
</tr>
<tr>
<td>0</td>
<td>The next random number in the sequence is selected</td>
</tr>
<tr>
<td>Otherwise</td>
<td>The generator is reseeded using NEW_SEED, then restarted, and the first random value is selected.</td>
</tr>
</tbody>
</table>

DRANDM is the synonym of DRAND. Both functions are included to insure portability of the existing code that reference either of them.

**Example**

```fortran
USE IFILPORT
REAL(8) num
INTEGER(4) f
f = 1
CALL print_rand
f = 0
CALL print_rand
f = 22
CONTAINS
SUBROUTINE print_rand
num = DRAND (f)
print *, 'f = ', f, ':', num
END SUBROUTINE
END
```
DRANSET

Sets the seed for RANGET

**Prototype**

```
INTERFACE
  SUBROUTINE DRANSET (ISEED)
  REAL(8) ISEED
  END SUBROUTINE DRANSET
END INTERFACE
```

**Description**

Allows you to set the seed for RANGET, changing the sequence of pseudo-random numbers.

**Output**

Changes the internal value of the random number generator seed for RANGET.

DSHIFTL

*Double shift left*

**Prototype**

```
INTERFACE
  INTEGER(8) FUNCTION DSHIFTL (LEFT, RIGHT, SHIFT)
  INTEGER(8) LEFT, RIGHT, SHIFT
  END FUNCTION DSHIFTL
END INTERFACE
```
LEFT a 64-bit integer expression.
RIGHT a 64-bit integer expression.
SHIFT shift count.

**Description**
This function takes two 64-bit values and a shift count, and returns a 64-bit value comprised of the 64 bits starting at 64-SHIFT in left continuing through to right.

**Example**

```fortran
INTERFACE
   INTEGER(8) FUNCTION DSHIFTL(LEFT,RIGHT,SHIFT)
   INTEGER(8) LEFT,RIGHT,SHIFT
   END FUNCTION DSHIFTL
END INTERFACE

   INTEGER(8) LEFT/Z'111122221111222'/
   INTEGER(8) RIGHT/Z'FFFFFFFFFFFFF'/
   PRINT *, DSHIFTL(LEFT, RIGHT, 16_8)
END
```

The correct output for this example is: 1306643199093243919.

**Output**
This function performs a shift left and extract, for compatibility with CRAY Fortran code.

---

**DSHIFTR**

*Double shift right*

**Prototype**

```fortran
INTERFACE
   INTEGER(8) FUNCTION DSHIFTR(LEFT,RIGHT,SHIFT)
END INTERFACE
```
INTEGER(8) LEFT, RIGHT, SHIFT
END FUNCTION DSHIFTR
END INTERFACE

ILEFT a 64-bit integer value.
IRIGHT a 64-bit integer value.
ISHIFT an integer shift count.

Description
Interprets ILEFT and IRIGHT as the upper and lower parts, respectively, of a 128-bit integer. The result is the 64-bit string beginning with the bit ISHIFT of the 128-bit integer. The arguments, ILEFT and IRIGHT, are not altered unless the function result is assigned to the same storage location as either ILEFT or IRIGHT. ISHIFT should be in the range 0 to 64, inclusive.

Example
INTERFACE
  INTEGER(8) FUNCTION DSHIFTR(LEFT, RIGHT, SHIFT)
    INTEGER(8) LEFT, RIGHT, SHIFT
  END FUNCTION DSHIFTR
END INTERFACE
  INTEGER(8) LEFT/Z'111122221111222'/
  INTEGER(8) RIGHT/Z'FFFFFFFFFFFFF'/
  PRINT *, DSHIFTR(LEFT, RIGHT, 16_8)
END

The correct output for this example is: 1306606910610341887.

Output
A single INTEGER(8) value.
DTIME

Returns the elapsed CPU time since the start of execution or the last call to DTIME.

Prototype

INTERFACE
  REAL(4) FUNCTION DTIME (TARRAY)
  REAL(4) TARRAY(2)
END FUNCTION DTIME
END INTERFACE

Description

On the first call during the execution of a program, DTIME returns the time since the beginning of the program execution in the elements of TARRAY. TARRAY (1) contains the user time, and TARRAY (2) contains the system time. The library routine makes use of the GetProcessTimes system library call. Subsequent calls to DTIME return the elapsed user and system time in TARRAY since the last call to DTIME.

Output

The function returns -1 for an error. The times are returned as elements of TARRAY, as described above. Times are in seconds. For the most accurate timing of your process, use the standard Fortran 95 intrinsic CPU_TIME.
**ETIME**

*Returns the elapsed time in seconds since the start of execution*

**Prototype**

```fortran
INTERFACE
   REAL(4) FUNCTION ETIME(TARRAY)
   REAL(4) TARRAY(2)
END FUNCTION
END INTERFACE
```

**Description**

This function returns the elapsed time in seconds since the beginning of execution of the current process. The accuracy of `ETIME` is limited to the accuracy of the `GetProcessTimes` system call, approximately 1/100th of a second. For the most accurate timing of your routine, use the Fortran 95 standard intrinsic `CPU_TIME`.

**Output**

The function returns -1 in case of an error. `TARRAY(1)` contains the user time in seconds since the start of the process. The system time in seconds since the start of the process is returned in `TARRAY(2)`.
EXIT
Closes all files and terminates the program

Prototype
INTERFACE
  SUBROUTINE EXIT (STATUS)
    INTEGER(4), OPTIONAL, INTENT(IN)::STATUS
  END SUBROUTINE
END INTERFACE

Optional Argument
STATUS

Description
Close all files and terminate the program.

Class
Nonstandard subroutine.

Argument
If STATUS is supplied, the calling program exits with a return code status of STATUS. Otherwise the return code status is indeterminate. In csh the $status environment variable holds the return code for the last executed command. In ksh, the $? environment variable holds the return code.

Example
The following program exits before the second PRINT statement.
PROGRAM testexit
  INTEGER stat
  PRINT *, "Program prints this line."
stat = 3
CALL EXIT(stat)
END
This program produces the following output:
Program prints this line.
The return code is saved in the $status or $? environment variable; it is not printed.

FDATE
Returns the current date and time

Prototype
INTERFACE
   SUBROUTINE FDATE(STRING)
      CHARACTER(LEN=24) STRING
   END SUBROUTINE FDATE
END INTERFACE
STRING a 24-byte character variable or array element in which the result of FDATE is stored.

Description
This routine returns the current date and time in STRING. Any value in STRING before the call is destroyed.

Output
A 24-character string with the form: Thu Jan 15 00:00:01 1970.
FGETC

Reads one byte from a file

Prototype

INTERFACE
    INTEGER FUNCTION FGETC(LUNIT,NCHAR)
    INTEGER LUNIT
    CHARACTER(LEN=1) NCHAR
END FUNCTION FGETC
END INTERFACE

LUNIT a logical unit number
NCHAR a character variable

Description

This routine reads one byte from a file at its current position. If there is an error during the read, the I/O error number is returned. Otherwise, for a successful read, FGETC returns zero. You should be aware that for unformatted files, special bytes such as a record length indicator are present on each record, and may require special handling when using this function. In general, record length indicators for unformatted files are the first four bytes of each record. The logical unit number must be in the range from 0 to 100, and must be currently connected to a file when FGETC is called.

This routine is thread-safe, and locks the associated stream before I/O is performed.

Output

A zero status is returned if successful, non-zero if failure. NCHAR is assigned the next sequential byte that would be read from the file connected to LUNIT.
**FINDFILEQQ**

*Searches for a specified file*

**Prototype**

```
USE IFLPORT
Or
INTERFACE
  INTEGER(4) FUNCTION FINDFILEQQ(FILE, ENV, BUF)
  CHARACTER(LEN=*) FILE, ENV, BUF
END FUNCTION
END INTERFACE
```

**Description**

Searches for a specified file in the directories listed in the path contained in the environment variable.

**Usage**

```
result = FINDFILEQQ (file, env, buf)
```

- **FILE** CHARACTER(LEN=*). Name of the file to be found.
- **ENV** CHARACTER(LEN=*). Name of an environment variable containing the path to be searched.
- **BUF** CHARACTER(LEN=*). Buffer to receive the full path of the file found.

**Results**

The result type is INTEGER(4). The result is the length of the characters containing the full path of the found file returned in BUF, or 0 if no file is found.

**Example**

```
USE IFLPORT
CHARACTER(256) BUF
```
CHARACTER(20) FILE, ENV
INTEGER(4) result
FILE = 'libc.lib'
ENV = 'LIB'
result = FINDFILEQQ(FILE, ENV, buf)
WRITE (*,*) BUF
END

flush
Flushes contentsoffilebuffertoan external file

Prototype
INTERFACE
  SUBROUTINE FLUSH(LUNIT)
    INTEGER LUNIT
  END SUBROUTINE FLUSH
END INTERFACE

LUNIT a logical unit number

Description
This routine flushes the contents of the file buffer to the external file, forcing an immediate write. This is most useful for files that are connected to a terminal. The logical unit number must be in the range from 0 to 100, and must be currently connected to a file when flush is called. This routine is thread-safe, and locks the associated stream before I/O is performed.

Output
ERRNO is set on failure.
**FOR_CHECK_FLAWED_PENTIUM**

*Checks the processor*

**Prototype**

```
USE IFLPORT
or
INTERFACE
    SUBROUTINE FOR_CHECK_FLAWED_PENTIUM
    END SUBROUTINE
END INTERFACE
```

**Description**

Checks the processor to determine if it shows characteristics of the Pentium® floating-point divide flaw.

It is invoked for a Fortran program if you compile with the /Qfdiv compiler switch.

**Usage**

```
result = FOR_CHECK_FLAWED_PENTIUM ( )
```

**Results**

If the floating-point divide flaw is found, an error message is displayed and the calling program is terminated.

You can bypass this action by setting environment variable `FOR_RUN_FLAWED_PENTIUM` to the value of `.TRUE.`

**Example**

```
USE IFLPORT
REAL*8 X, Y, Z
X = 5244795.0
Y = 3932159.0
```
Z = X - (X/Y) * Y
IF (Z .NE. 0) THEN ! If flawed, Z will be 256
    PRINT *, ' FDIV flaw detected on Pentium'
ENDIF
END

FOR_GET_FPE

Return the current settings of the floating-point exception flags

Prototype

USE IFLPORT
or

INTERFACE
    INTEGER(4) FUNCTION FOR_GET_FPE()
    END FUNCTION
END INTERFACE

Description

Returns the current settings of floating-point exception flags. This routine can be called from a C or Fortran program.

Usage

result = FOR_GET_FPE ( )

Results

The result type is INTEGER(4). The return value represents the settings of the current processor floating-point exception flags. The meanings of the bits are defined in the IFLPORT module file.
Example
USE IFLPORT
INTEGER(4) FPE_FLAGS
FPE_FLAGS = FOR_GET_FPE ( )
END

FPUTC
*Writes a character a a file*

Prototype

INTERFACE
   INTEGER FUNCTION FPUTC(LUNIT, CH)
   INTEGER LUNIT
   CHARACTER(LEN=1) CH
END FUNCTION FPUTC
END INTERFACE

LUNIT     unit number of a file.
CH        a character variable.

Description
Writes a character to the file specified by the Fortran external unit.

Output
Zero if successful. Otherwise, an error code is returned.
FSEEK

Positions a file from a specified seek point

Prototype

INTERFACE
  SUBROUTINE FSEEK(LUNIT, OFFSET, FROM)
  INTEGER(4) LUNIT, OFFSET, FROM
  END SUBROUTINE FSEEK
END INTERFACE

LUNIT a logical unit number.
OFFSET an integer expression, whose value denotes an offset in bytes from the seek point FROM.
FROM an integer expression, whose value must be 0, 1, or 2.
0 means the seek point is at the beginning of the file.
1 means the seek point is on the current file position.
2 means the seek point is at the end of the file.

Description

This routine positions a file OFFSET bytes from the seek point given by the FROM parameter. You should be aware that for unformatted files, special bytes such as a record length indicator are present on each record, and may require special handling when using this function. In general, record length indicators for unformatted files are the first four bytes of each record. The logical unit number must be in the range from 0 to 100, and must be currently connected to a file when fseek is called.

This routine is thread-safe, and locks the associated stream before I/O is performed.
**Output**

A zero status is returned if successful; non-zero for failure.

---

**FOR_SET_FPE**

*Sets the floating-point exception flags.*

**Prototype**

```fortran
USE IFLPORT
or
INTERFACE
  INTEGER(4) FUNCTION FOR_SET_FPE(EnableMask)
  INTEGER(4) EnableMask
  END FUNCTION
END INTERFACE
```

**Usage**

```fortran
result = FOR_SET_FPE(EnableMask)
EnableMask Must be of type INTEGER(4). It contains bit flags controlling floating-point exceptions.
```

**Results**

The result type is INTEGER(4). The return value represents the previous settings of the floating-point exception flags. The meanings of the bits are defined in the IFLPORT.f90 module file.

**Example**

```fortran
USE IFLPORT
INTEGER(4) OLD_FLAGS, NEW_FLAGS
OLD_FLAGS = FOR_SET_FPE (NEW_FLAGS)
END
```
FOR_SET_REENTRANCY

Controls the type of reentrance locks on the runtime library.

Prototype

USE IFLPORT
or

INTERFACE
  INTEGER(4) FUNCTION FORr_SET_REENTRANCY(NEW_MODE)
    INTEGER(4) NEW_MODE
  END FUNCTION
END INTERFACE

Usage

result = FOR_SET_REENTRANCY(NEW_MODE)

NEW_MODE INTEGER(4). Contains one of the following options:

FOR_K_REENTRANCY_NONE
  Tells the runtime to do simple locking around critical sections of RTL code. You should use this type of protection when the Fortran libraries will not be reentered due to asynchronous system traps (ASTs) or threads within the application.

FOR_K_REENTRANCYASYNCH
  Tells the runtime to perform simple locking and disables ASTs around critical sections of library code. You should use this type of protection when the application contains AST handlers that call the Fortran runtime system.

FOR_K_REENTRANCY_THREADED
  Tells the runtime to perform thread locking. You should use this type of protection in multithreaded applications.
FOR_K_REENTRANCY_INFO
Queries the Fortran runtime system for the current level
of reentrance protection, and returns the result in
NEW_MODE.

Results
FOR_SET_REENTRANCY returns INTEGER(4). The return value tells you
the previous setting of reentrancy NEW_MODE, unless the argument is
FOR_K_REENTRANCY_INFO, in which case the return value represents the
current setting.

You must link to a set of runtime libraries that support the level of
reentrance you desire. For example, FOR_SET_REENTRANCY ignores a
request for thread protection (FOR_K_REENTRANCY_THREADED) if you do
not compile your program with /MT.

Example
PROGRAM SETREENT
  USE IFLPORT
  INTEGER(4) MODE
  CHARACTER*10 REENT_TXT(3) /'NONE ', 'ASYNCH ',
  'THREADED'/
  INTEGER(4) P1, P2
  P1 = FOR_K_REENTRANCY_NONE
  P2 = FOR_K_REENTRANCY_INFO
  PRINT*, 'Setting Reentrancy mode to '
  'REENT_TXT(MODE+1)'
  MODE = FOR_SET_REENTRANCY(P1)
  PRINT*, 'Previous Reentrancy mode was '
  'REENT_TXT(MODE+1)
  MODE = FOR_SET_REENTRANCY(P2)
  PRINT*, 'Current Reentrancy mode is '
  'REENT_TXT(MODE+1)
END
FSTAT

*Return detailed information about a file specified by external unit number.*

**Prototype**

```fortran
USE IFLPORT
```

**Usage**

```fortran
result = FSTAT (LUNIT, STATB)
```

- **LUNIT**
  - Input. INTEGER (4). External unit number of the file to examine.

- **STATB**
  - Output. INTEGER (4). One-dimensional array with the size of 12. The following table lists the elements of the array.

<table>
<thead>
<tr>
<th>STATB</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATB(1)</td>
<td>Device the file resides on (always 0)</td>
</tr>
<tr>
<td>STATB(2)</td>
<td>Inode number (always 0)</td>
</tr>
<tr>
<td>STATB(3)</td>
<td>File type, attribute, and access control information (see the next table for this function)</td>
</tr>
<tr>
<td>STATB(4)</td>
<td>Number of links (always 1)</td>
</tr>
<tr>
<td>STATB(5)</td>
<td>User ID of owner (always 1)</td>
</tr>
<tr>
<td>STATB(6)</td>
<td>Group ID of owner (always 1)</td>
</tr>
<tr>
<td>STATB(7)</td>
<td>Raw device the file resides on (always 1)</td>
</tr>
<tr>
<td>STATB(8)</td>
<td>The size of the file in bytes</td>
</tr>
<tr>
<td>STATB(9)</td>
<td>The time of last access (only available on non-FAT file systems; same as STATB(10) on FAT systems.</td>
</tr>
<tr>
<td>STATB(10)</td>
<td>The time of last modification</td>
</tr>
<tr>
<td>STATB(11)</td>
<td>The time of last status change (same as STATB(10))</td>
</tr>
<tr>
<td>STATB(12)</td>
<td>Block size (always 1)</td>
</tr>
</tbody>
</table>
Results

The result is of type \texttt{INTEGER(4)}. The result is zero if successful; otherwise returns an error code equal to \texttt{EUNVAL} (\texttt{LUNIT} is not a valid unit number or is not open).

Mode is a bitmap consisting of the IOR of the following constants (the module \texttt{IFLPORT} supplies parameters with the symbolic names given).

<table>
<thead>
<tr>
<th>Symbolic name</th>
<th>Constant</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IFMT</td>
<td>O'0170000'</td>
<td>Type of file</td>
<td></td>
</tr>
<tr>
<td>S_IFDIR</td>
<td>O'0040000'</td>
<td>Directory</td>
<td></td>
</tr>
<tr>
<td>S_IFCHR</td>
<td>O'0020000'</td>
<td>Character special</td>
<td>Never set</td>
</tr>
<tr>
<td>S_IFBLK</td>
<td>O'0060000'</td>
<td>Block special</td>
<td>Never set</td>
</tr>
<tr>
<td>S_IFREG</td>
<td>O'0100000'</td>
<td>Regular</td>
<td></td>
</tr>
<tr>
<td>S_IFLNK</td>
<td>O'0120000'</td>
<td>Symbolic link</td>
<td>Never set</td>
</tr>
<tr>
<td>S_IFSOCK</td>
<td>O'0140000'</td>
<td>Socket</td>
<td>Never set</td>
</tr>
<tr>
<td>S_ISUID</td>
<td>O'0004000'</td>
<td>Set user ID on execution</td>
<td>Never set</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>O'0002000'</td>
<td>Set group ID on execution</td>
<td>Never set</td>
</tr>
<tr>
<td>S_ISVTX</td>
<td>O'0001000'</td>
<td>Save swapped text</td>
<td>Never set</td>
</tr>
<tr>
<td>S_IRWXU</td>
<td>O'0000700'</td>
<td>Owner's file permission</td>
<td></td>
</tr>
<tr>
<td>S_IRUSR,</td>
<td>O'0000400'</td>
<td>Owner read permission</td>
<td>Always true</td>
</tr>
<tr>
<td>S_IWUSR,</td>
<td>O'0000200'</td>
<td>Owner write permission</td>
<td></td>
</tr>
<tr>
<td>S_IWRITE</td>
<td>O'0000100'</td>
<td>Owner execute permission</td>
<td>Set if \texttt{S_IREAD} is set</td>
</tr>
<tr>
<td>S_IXUSR,</td>
<td>O'0000100'</td>
<td>Owner execute permission</td>
<td></td>
</tr>
<tr>
<td>S_IEXEC</td>
<td>O'0000070'</td>
<td>Group's life permissions</td>
<td>Same as \texttt{S_IRWXU}</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>O'0000040'</td>
<td>Group read permission</td>
<td>Same as \texttt{S_IRUSR}</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>O'0000020'</td>
<td>Group write permission</td>
<td>Same as \texttt{S_IWUSR}</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>O'0000010'</td>
<td>Group execute permission</td>
<td>Same as \texttt{S_IXUSR}</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>O'0000007'</td>
<td>Other's file permissions</td>
<td>Same as \texttt{S_IRWXU}</td>
</tr>
</tbody>
</table>
Time values are returned as number of seconds since 0:00:00 GMT, January 1, 1970.

Example
USE IFLPORT
INTEGER(4) STATARRAY(12), ISTAT
OPEN (UNIT=1, FILE='DATAFILE.DAT')
ISTAT = FSTAT(1, STATARRAY)
IF (.NOT. ISTAT) THEN
  PRINT *, STATARRAY
ENDIF

FTELL
Returns the file position of a file

Prototype
INTERFACE
  INTEGER(4) FUNCTION FTELL(LUNIT)
END FUNCTION FTELL
END INTERFACE

LUNIT a logical unit number

Description
This routine returns the file position of the file connected to the input logical unit. The file position is the number of bytes from the beginning of the file.
You should be aware that for unformatted files, special bytes such as a record length indicator are present on each record, and are counted when using this function. In general, record length indicators for unformatted files are the first four bytes of each record.

The logical unit number must be in the range from 0 to 100, and must be currently connected to a file when ftell is called.

This routine is thread-safe, and locks the associated stream before I/O is performed.

**Output**

ERRNO is set on failure.

---

**FULLPATHQQ**

*Returns the full path for a specified file or directory.*

**Prototype**

USE IFLPORT
or
INTERFACE
    INTEGER(4) FUNCTION FULLPATHQQ(NAME, FULLPATH)
    CHARACTER(LEN=*) NAME, FULLPATH
END FUNCTION
END INTERFACE

**Usage**

result = FULLPATHQQ ( name, fullpath )

NAME CHARACTER(LEN=*). File name for which you want a full path. The file can be the name of a file in the current directory, or a relative directory or filename.
FULLPATH CHARACTER(LEN=*). CHARACTER VALUE that receives the full path of the item specified in NAME.

Results
The result is the length of the full pathname in CHARACTERS, or 0 if the function fails. The function may fail if the name supplied is not an existing file. The length of FULLPATH will vary, depending on how deeply the directories are nested on the drive you are using. If the full path is longer than the character buffer provided to return it (FULLPATH), FULLPATHQQ returns only that portion of the path that fits.

You should verify the length of the path before using the value returned in FULLPATH. If the longest full path you are likely to encounter does not fit into the buffer you are using, allocate a larger character buffer. You can allocate the largest possible path buffer with the following statements:

USE IFLPORT
CHARACTER(MAXPATH) FULLPATH

MAXPATH is a symbolic constant defined in module IFLPORT.F90 as 260.

Example
USE IFLPORT
CHARACTER(MAXPATH) BUF
CHARACTER(3) DRIVE
CHARACTER(256) DIR
CHARACTER(256) NAME
CHARACTER(256) EXT
CHARACTER(256) FILE
INTEGER(4) LEN

DO WHILE (.TRUE.)
    WRITE (*,*)
    WRITE (*,'(A)') ' Enter filename (Hit RETURN to exit): '
    LEN = GETSTRQQ(FILE)
    IF (LEN .EQ. 0) EXIT
    LEN = FULLPATHQQ(FILE, BUF)
IF (LEN .GT. 0) THEN
   WRITE (*,*) buf(:len)
ELSE
   WRITE (*,*) 'Can''t get full path'
   EXIT
END IF

! Split path
WRITE (*,*)
LEN = SPLITPATHQQ(BUF, DRIVE, DIR, NAME, EXT)
IF (LEN .NE. 0) THEN
   WRITE (*, 900) ' Drive: ', DRIVE
   WRITE (*, 900) ' Directory: ', DIR(1:LEN)
   WRITE (*, 900) ' Name: ', NAME
   WRITE (*, 900) ' Extension: ', EXT
ELSE
   WRITE (*, *) 'Can''t split path'
END IF
END DO
900 FORMAT (A, A)
END

GERROR

Returns a message for last error

Prototype

INTERFACE
   SUBROUTINE GERROR(ERRORMSG)
      CHARACTER(LEN=*) ERRORMSG
   END SUBROUTINE
END INTERFACE
STRING message for the last error detected

Description
This routine returns a message for the last error detected.

Output
The last error detected in STRING

GETARG

Gets a specified command-line argument

Prototype
INTERFACE GETARG
   SUBROUTINE GETARG_DVF(N, BUFFER, STATUS)
      INTEGER(2), INTENT(IN) :: N
      CHARACTER(LEN=*) , INTENT(OUT) :: BUFFER
      INTEGER(2), OPTIONAL :: STATUS
   END SUBROUTINE
   SUBROUTINE GETARG(ARGINDEX, ARGUMENT)
      INTEGER(4), INTENT(IN) :: ARGINDEX
      CHARACTER(LEN=*) , INTENT(OUT) :: ARGUMENT
   END SUBROUTINE
END INTERFACE

N (input) INTEGER(2). Position of the command-line argument to retrieve. The command itself is argument number 0.

BUFFER (output) the space for argument to be returned into. Must be large enough, otherwise the result is truncated.

STATUS (optional; output) INTEGER(2). STATUS is the original number of characters in the command-line argument. If specified, returns the completion status, that is, the number of characters in the retrieved command-line.
argument before truncation or blank-padding. Errors return a value of -1. Errors include specifying an argument position less than 0 or greater than the value returned by NARGS.

Description
GETARG returns the Nth command-line argument. If N is zero, GETARG returns the name of the executing program file.

GETARG can be used with two or three arguments. If you use module DFLIB.F90, you can mix calls to GETARG with two or three arguments, see Example below. If you do not use DFLIB.F90, you can use either two- or three-argument calls to GETARG but only one type within a subprogram.

With two arguments, the first one is of type INTEGER(KIND=4). This version simply returns the Nth argument as a character variable. ARGINDEX (or N) must be zero or a positive integer. If there is an error while retrieving the argument, ARGUMENT (or BUFFER) is filled with blanks. If ARGUMENT is too short to hold the input argument, the input argument is truncated to the length corresponding to ARGUMENT. If ARGUMENT is longer than the input argument, then ARGUMENT is blank-filled on the right.

The second form of GETARG allows an optional parameter for the length of the input argument returned, before blank-padding or truncation. It also uses an INTEGER(KIND=2) argument index.

Output
A string representing the Nth command-line argument to the executing program as it was entered. There is no case conversion

Example
The following code illustrates a mix call to GETARG.

USE DFLIB
INTEGER(2) result
result = RUNQQ('prog', '-c -r')
END
! PROG.F90
USE DFLIB
INTEGER(2) n1, n2, status
GETC

Reads the next available character from external unit 5, which is normally connected to the console.

Prototype

USE IFLPORT

Usage

result = GETC (CHAR)

CHAR Output. CHARACTER (LEN=*) . First character typed at the keyboard after the call to GETC. If unit 5 is connected to a console device, then no characters are returned until the Enter key is pressed.

Results

The result is of type INTEGER (4) . The result is zero if successful, or -1 if end-of-file was detected.

Example

USE IFLPORT
CHARACTER ANS, ERRTXT*40
PRINT *, ' Enter a character: '
ISTAT = GETC (ANS)
IF (ISTAT) THEN
CALL GERROR (ERRTXT)
ENDIF

GETCHARQQ

Gets the next keystroke.

Prototype
USE IFLPORT
or
! Get character from console
CHARACTER(LEN=1) FUNCTION GETCHARQQ()
END FUNCTION GETCHARQQ

Usage
result = GETCHARQQ ( )

Results
The result type is CHARACTER (1), and has the value of the key that was pressed. The value can be any ASCII character. If the key pressed is represented by a single ASCII character, GETCHARQQ returns the character. If the key pressed is a function or direction key, a hex #00 or #E0 is returned. If you need to know which function or direction was pressed, call GETCHARQQ a second time to get the extended code for the key.
If there is no keystroke waiting in the keyboard buffer, GETCHARQQ will wait indefinitely until there is one, and then returns it. To determine in advance whether there is a character in the keyboard buffer, use PEEKCHARQQ, which returns .TRUE. if there is a character waiting in the keyboard buffer, and .FALSE. if not. This can prevent a program from hanging while GETCHARQQ waits for a keystroke that isn't there.

Example
! Program to demonstrate GETCHARQQ
USE IFLPORT
CHARACTER(1) key / 'A' / 
PARAMETER (ESC = 27)
PARAMETER (NOREP = 0)
WRITE (*,*) ' Type a key: (or q to quit)'
! Read keys until ESC or q is pressed
DO WHILE (ICHAR (key) .NE. ESC)
   key = GETCHARQQ()
   ! Some extended keys have no ASCII representation
   IF (ICHAR(key) .EQ. NOREP) THEN
      key = GETCHARQQ()
      WRITE (*, 900) 'Not ASCII. Char = NA'
      WRITE (*,*)
   ELSE
      WRITE (*,900) 'ASCII. Char = '
      WRITE (*,901) key
   END IF
   IF (key .EQ. 'q' ) THEN
      EXIT
   END IF
END DO
900 FORMAT (1X, A)
901 FORMAT (A)
END

GETCONTROLFPQQ

Returns the floating-point processor control word.

Prototype

USE IFLPORT

or

INTERFACE
   SUBROUTINE GETCONTROLFPQQ(CONTROL)
INTEGER(2) CONTROL
END SUBROUTINE
END INTERFACE

Usage
CALL GETCONTROLFPQQ (CONTROL)
CONTROL INTEGER(2). Floating-point processor control word.
The floating-point control word is a set of flags that determines various
modes of the floating-point co-processor. The IFLPORT.F90 module file
contains constants defined for the control word as follows:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPCW$MCW_IC</td>
<td>Z‘1000’</td>
<td>Infinity control mask</td>
</tr>
<tr>
<td>FPCW$AFFINE</td>
<td>Z‘1000’</td>
<td>Affine infinity</td>
</tr>
<tr>
<td>FPCW$PROJECTIVE</td>
<td>Z‘0000’</td>
<td>Projective infinity</td>
</tr>
<tr>
<td>FPCW$MCW_PC</td>
<td>Z‘0300’</td>
<td>Precision control mask</td>
</tr>
<tr>
<td>FPCW$64</td>
<td>Z‘0300’</td>
<td>64-bit precision</td>
</tr>
<tr>
<td>FPCW$53</td>
<td>Z‘0200’</td>
<td>53-bit precision</td>
</tr>
<tr>
<td>FPCW$24</td>
<td>Z‘0000’</td>
<td>24-bit precision</td>
</tr>
<tr>
<td>FPCW$MCW_RC</td>
<td>Z‘0C00’</td>
<td>Rounding control mask</td>
</tr>
<tr>
<td>FPCW$CHOP</td>
<td>Z‘0C00’</td>
<td>Truncate</td>
</tr>
<tr>
<td>FPCW$UP</td>
<td>Z‘0800’</td>
<td>Round up</td>
</tr>
<tr>
<td>FPCW$DOWN</td>
<td>Z‘0400’</td>
<td>Round down</td>
</tr>
<tr>
<td>FPCW$NEAR</td>
<td>Z‘0000’</td>
<td>Round to nearest</td>
</tr>
<tr>
<td>FPCW$MSW_EM</td>
<td>Z‘003F’</td>
<td>Exception mask</td>
</tr>
<tr>
<td>FPCW$INVALID</td>
<td>Z‘0001’</td>
<td>Allow invalid numbers</td>
</tr>
<tr>
<td>FPCW$DENORMAL</td>
<td>Z‘0002’</td>
<td>Allow denormals (very small numbers)</td>
</tr>
<tr>
<td>FPCW$ZERODIVIDE</td>
<td>Z‘0004’</td>
<td>Allow divide by zero</td>
</tr>
<tr>
<td>FPCW$OVERFLOW</td>
<td>Z‘0008’</td>
<td>Allow overflow</td>
</tr>
<tr>
<td>FPCW$UNDERFLOW</td>
<td>Z‘0010’</td>
<td>Allow underflow</td>
</tr>
<tr>
<td>FPCW$INEXACT</td>
<td>Z‘0020’</td>
<td>Allow inexact precision</td>
</tr>
</tbody>
</table>
By default, the floating-point control word settings are 53-bit precision, round to nearest, and the DENORMAL, UNDERFLOW and INEXACT precision exceptions disabled. An exception is disabled if its flag is set to 1 and enabled if its flag is cleared to 0. You can disable exceptions by setting the flags to 1 with SETCONTROLFPQQ.

If an exception is disabled, it does not cause an interrupt when it occurs. Instead, the exception generates an appropriate special value (NaN or signed infinity), but the program continues. When printing out such a value, the runtime library represents a NaN as ?????, positive infinity as ++++++, and negative infinity as .......... 

You can find out which exceptions (if any) occurred by calling GETSTATUSFPQQ. If you have enabled errors on floating-point exceptions, clearing the flags to 0 with SETCONTROLFPQQ, an interrupt is generated when the exception occurs. Normally, these interrupts cause errors, but you can capture the interrupts with SIGNALQQ and branch to your own error-handling routines.

You can use GETCONTROLFPQQ to retrieve the current control word and SETCONTROLFPQQ to change the control word. In most cases, you will not need to change the default settings.

Example

USE IFLPORT
INTEGER(2) CONTROL
CALL GETCONTROLFPQQ (CONTROL)
PRINT 10,CONTROL
10 FORMAT('Initial control settings ',Z)
   !if not rounding down
   IF (IAND(CONTROL, FPCW$DOWN) .NE. FPCW$DOWN) THEN
      CONTROL = IAND(CONTROL, NOT(FPCW$MCW_RC))
      !clear all rounding
   CONTROL = IOR(control, FPCW$DOWN)
      !set to round down
   CALL SETCONTROLFPQQ(CONTROL)
   CALL GETCONTROLFPQQ(CONTROL)
   PRINT 20,control
20 FORMAT('Final control settings ',Z)
END IF
END
**GETCWD**

Retrieves the path of the current working directory.

---

**Prototype**

USE IFLPORT

or

INTERFACE

   INTEGER(4) FUNCTION GETCWD(DIRECTORY)
   CHARACTER(LEN=*) DIRECTORY

END FUNCTION GETCWD

END INTERFACE

**Usage**

result = GETCWD(directory)

DIRECTORY CHARACTER (LEN=*). Character value that receives the current working directory path, including drive letter.

**Results**

GETCWD returns zero for success, or an error code for failure.

**Example**

USE IFLPORT

CHARACTER(LEN=30) DIRECTORY

! variable DIRECTORY must be long enough to hold entire string

INTEGER(4) ISTAT

ISTAT = GETCWD(DIRECTORY)

IF (ISTAT == 0) PRINT *, 'Current directory is ', DIRECTORY

END
GETDAT

*Returns the current date in integer form*

**Prototype**

```fortran
INTERFACE GETDAT
  SUBROUTINE GETDAT(IYEAR,IMONTH,IDAY)
    INTEGER(4), INTENT(OUT) :: IYEAR,IMONTH,IDAY
  END SUBROUTINE
  SUBROUTINE GETDAT_DVF(IYEAR,IMONTH,IDAY)
    INTEGER(2), INTENT(OUT) :: IYEAR,IMONTH,IDAY
  END SUBROUTINE
END INTERFACE
```

**IYEAR**
an integer value

**IMONTH**
an integer value

**IDAY**
an integer value

**Description**

This routine returns the current date in integer form. On Windows NT* systems, this function is thread-safe.

**Output**

The current date is returned. **IYEAR** contains the current year, reckoned by the Julian calendar. **IMONTH** contains the numerical version of the month, where 1 corresponds to January, and 12 corresponds to December. **IDAY** returns the numerical day of the month.
GETDRIVEDIRQQ

Gets the complete path of the current working directory on a specified disk drive.

Prototype

USE IFLPORT
or INTERFACE

! Get the current directory for a given drive
INTEGER(4) FUNCTION GETDRIVEDIRQQ(DRIVEDIR)
CHARACTER(LEN=*) DRIVEDIR
END FUNCTION
END INTERFACE

Usage

result = GETDRIVEDIRQQ (DRIVEDIR)

DRIVEDIR CHARACTER(LEN=*) For input, DRIVEDIR contains the drive whose current working directory path is to be returned. On output, DRIVEDIR contains the current directory on that drive in the form d:\dir.

Results

The result is INTEGER(4). The result is the length (in bytes) of the full path on the specified drive. If the full path is longer than the size of DRIVEDIR, zero is returned.

You can make sure you get information about the current drive, by putting the symbolic constant FILE$CURDRIVE (defined in IFLPORT.F90) into DRIVEDIR.

Since disk drives are identified by a single alphabetic character, GETDRIVEDIRQQ examines only the first letter of DRIVEDIR. For instance, if DRIVEDIR contains the path c:\program files,
GETDRIVEDIRQQ (DRIVEDIR) returns the current working directory on drive C and disregards the rest of the path. Input is case-insensitive. The length of the path returned depends on how deeply the directories are nested on the drive specified in DRIVEDIR. If the full path is longer than the length of DRIVEDIR, GETDRIVEDIRQQ returns only the portion of the path that fits into DRIVEDIR. If you are likely to encounter a long path, allocate a buffer of size MAXPATH (where MAXPATH is a PARAMETER constant defined in IFLPORT.F90, and MAXPATH = 260).

On Linux platforms, the function gets a path only when symbolic constant FILE$CURDRIVE (defined in IFLPORT.F90) is applied for argument drivedir.

Example

! Program to demonstrate GETDRIVEDIRQQ
USE IFLPORT
CHARACTER(MAXPATH) dir
INTEGER(4) length
! Get current directory
dir = FILE$CURDRIVE
length = GETDRIVEDIRQQ(dir)
IF (length .GT. 0) THEN
  WRITE (*,*) 'Current directory is: '
  WRITE (*,*) dir
ELSE
  WRITE (*,*) 'Failed to get current directory'
END IF
END
GETDRIVESIZEQQ

*Gets the total size of the specified DRIVE.*

**Prototype**

```fortran
USE IFLPORT
or
INTERFACE GETDRIVESIZEQQ
   LOGICAL(4) FUNCTION GETDRIVESIZEQQI4 (DriveNm, TotalNum, AvailableNum)
      CHARACTER(LEN=*) DriveNm
      INTEGER(4) TotalNum
      INTEGER(4) AvailableNum
   END FUNCTION
   LOGICAL(4) FUNCTION GETDRIVESIZEI8 (DriveNm, TotalNum, AvailableNum)
      CHARACTER(LEN=*) DriveNm
      INTEGER(8) TotalNum
      INTEGER(8) AvailableNum
   END FUNCTION
END INTERFACE
```

**Description**

Gets the total size of the specified DRIVE and space available on it.

**Usage**

```fortran
result = GETDRIVESIZEQQ (DRIVENm, TOTALNUM, & AVAILIABLENUM)
```

- **DriveNm**
  - CHARACTER(LEN=*). String containing the letter of the disk drive to get information about.

- **TotalNum**
  - INTEGER(4) or INTEGER(8). TotalNum number of bytes on the disk drive
AvailableNum INTEGER(4) or INTEGER(8). Number of bytes of AVAILable space on the disk drive.

Results
The function returns LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

The data types specified for the TotalNum and AvailableNum arguments must be the same.

Some disk drives have more bytes than will fit into a 32 bit integer. Using an INTEGER(4) variable for these drives will return a negative size. To get the actual size for these large drives, TotalNum and AvailableNum must be INTEGER(8) variables.

Because disk drives are identified by a single alphabetic character, GETDRIVESIZEQQ examines only the first letter of DriveNm. The drive letter can be uppercase or lowercase. You can use the constant FILE$CURDRIVE (defined in IFLPORT.F90) to get the size of the current DRIVE.

If GETDRIVESIZEQQ fails, use GETLASTERRORQQ to determine the reason.

On Linux platforms, this routine always returns .false..

Example
! Program to demonstrate GETDRIVESQQ and GETDRIVESIZEQQ
USE IFLPORT
CHARACTER(26) drives
CHARACTER(1) adrive
LOGICAL(4) status
INTEGER(4) total, avail
INTEGER(2) i
! Get the list of drives
drives = GETDRIVESQQ()
WRITE (*, '(A, A)') ' Drives available: ', drives
!
!Cycle through them for free space and write to console
DO i = 1, 26
   adrive = drives(i:i)
   status = .FALSE.
   WRITE (*, '(A, A, A, )') ' Drive ', CHAR(i + 64), ':'
   IF (adrive .NE. ' ') THEN
      status = GETDRIVESIZEQQ(adrive, total, avail)
   END IF
   IF (status) THEN
      WRITE (*,*) avail, ' of ', total, ' bytes free.'
   ELSE
      WRITE (*,*) 'Not available'
   END IF
END DO
END

GETDRIVESQQ

Reports which drives are available to the system.

Prototype
USE IFLPORT
or
INTERFACE
   CHARACTER(26) FUNCTION GETDRIVESQQ()
   END FUNCTION
END INTERFACE

Usage
result = GETDRIVESQQ ()
Results
The result is CHARACTER (LEN=26). The returned string contains letters for drives that are available, and blanks for drives that are not available. For example, on a system with A, C, and D drives, the string 'A CD' is returned.

On Linux platforms, this function returns a string filled with spaces.

GETENV
Return the value of a system environment variable.

Prototype

INTERFACE
  SUBROUTINE GETENV (VAR, VALUE)
    CHARACTER(LEN=*) VAR, VALUE
  END SUBROUTINE
END INTERFACE

VAR must be CHARACTER type. Specifies the environment variable name.

VALUE must be CHARACTER type. The VALUE is assigned the environment variable’s value. VALUE must be declared large enough to hold the value. If the environment variable is not defined, VALUE is set to all blanks.

Description
Returns the value of a system environment variable.

Class
Nonstandard subroutine.
Example
The following code assigns VAL the value of the TERM environment variable.

CHARACTER(10) VAL
CALL GETENV('TERM', VAL)

GETENVVQQ

Gets the value of a specified environment variable.

Prototype
USE IFLPORT
or
! ALTERNATIVE WAY OF GETTING ENVIRONMENT VARIABLE VALUE
INTERFACE
  INTEGER(4) FUNCTION GETENVVQQ(VARNAME, VALUE)
  CHARACTER(LEN=*) VARNAME, VALUE
END FUNCTION GETENVVQQ
END INTERFACE

Description
Gets the value of a specified environment variable from the current environment.

Usage
result = GETENVVQQ (VARNAME, VALUE)

VARNAME                 CHARACTER(LEN=*). The name of environment variable whose value you want to find.
VALUE                   CHARACTER(LEN=*). Value of the specified environment variable, in uppercase.
Results

The result type is INTEGER(4). The result is the number of characters returned in VALUE. Zero is returned if the given variable is not defined.

GETENVQQ searches the list of environment variables for an entry corresponding to VARNAME. Environment variables define the environment in which a process executes. For example, the LIB environment variable defines the default search path for libraries to be linked with a program. Note that some environment variables may exist only on a per-process basis, and may not be present at the command-line level.

Example

! Program to demonstrate GETENVQQ and SETENVQQ
USE IFLPORT
INTEGER(4) lenv, lval
CHARACTER(80) env, val, enval
WRITE (*,900) ' Enter environment variable name to create, modify, or delete: '
  lenv = GETSTRQQ(env)
  IF (lenv .EQ. 0) STOP
  WRITE (*,900) 'Value of variable (ENTER to delete): '
  lval = GETSTRQQ(val)
  IF (lval .EQ. 0) val = ' '
  enval = env(1:lenv) // '=' // val(1:lval)
  IF (SETENVQQ(enval)) THEN
    lval = GETENVQQ(env(1:lenv), val)
    IF (lval .EQ. 0) THEN
      WRITE (*,*) 'Can''t get environment variable'
    ELSE IF (lval .GT. LEN(val)) THEN
      WRITE (*,*) 'Buffer too small'
    ELSE
      WRITE (*,*) env(:lenv), ': ', val(:lval)
      WRITE (*,*) 'Length: ', lval
    END IF
  ELSE
    WRITE (*,*) 'Can''t set environment variable'
GETFILEINFOQQ

Returns information about the specified file.

Prototype

IA-32 systems

USE IFLPORT or INTERFACE

INTEGER(4) FUNCTION GETFILEINFOQQ(FILES, BUFFER, DWHANDLE)

CHARACTER(LEN=*) FILES
TYPE FILEINFO
SEQUENCE
INTEGER(4) CREATION ! CREATION TIME (-1 ON FAT)
INTEGER(4) LASTWRITE ! LAST WRITE TO FILE
INTEGER(4) LASTACCESS ! LAST ACCESS (-1 ON FAT)
INTEGER(4) LENGTH ! LENGTH OF FILE
INTEGER(2) PERMIT ! FILE ACCESS MODE
CHARACTER(LEN=255) NAME ! FILE NAME
END TYPE

TYPE(FILEINFO) :: BUFFER

INTEGER(4) DWHANDLE

END FUNCTION GETFILEINFOQQ

END INTERFACE
Itanium®-based systems

INTERFACE
INTEGER(8) FUNCTION GETFILEINFOQQ(FILES,BUFFER, DWHANDLE)

  CHARACTER(LEN=*) FILES
  TYPE FILEINFO
  SEQUENCE
    INTEGER(8) CREATION ! Creation time (-1 ON FAT)
    INTEGER(8) LASTWRITE ! LAST WRITE TO FILE
    INTEGER(8) LASTACCESS ! LAST ACCESS (-1 ON FAT)
    INTEGER(4) LENGTH ! LENGTH OF FILE
    INTEGER(2) PERMIT ! FILE ACCESS MODE
    CHARACTER(LEN=255) NAME ! FILE NAME
  END TYPE
  TYPE(FILEINFO) :: BUFFER
  INTEGER(8) DWHANDLE
END FUNCTION GETFILEINFOQQ
END INTERFACE

Description

Returns information about the specified file. Filenames can contain wildcards (* and ?).

Usage

result = GETFILEINFOQQ (FILES,BUFFER, HANDLE)

FILES CHARACTER(LEN=*): Name or pattern of files that you are looking for information on. It can include a full path and can include wildcards (* and ?).

BUFFER Derived type FILEINFO. Information about a file that matches the search criteria is returned in this variable.

HANDLE INTEGER(4). Control mechanism. One of the following constants, defined in IFLPORT.F90:
FILEFIRST: First matching file found.
FILELAST: Previous file was the last valid file.
FILE$ERROR: No matching file found.

Results
The result is INTEGER(4), showing the nonblank length of the filename if a match was found, or 0 if no matching FILES were found.

To get information about one or more files, set HANDLE to FILE$FIRST and call GETFILEINFOQQ. This will return information about the first file which matches the name and return a HANDLE. If the program wants more files, it should call GETFILEINFOQQ with the HANDLE. GETFILEINFOQQ must be called with the HANDLE until GETFILEINFOQQ sets HANDLE to FILE$LAST, or system resources may be lost.

The derived-type element variables FILE$INFO%CREATION, FILE$INFO%LASTWRITE, and FILE$INFO%LASTACCESS contain packed date and time information that indicates when the file was created, last written to, and last accessed, respectively. To break the time and date into component parts, call UNPACKTIMEQQ. FILE$INFO%LENGTH contains the length of the file in bytes. FILE$INFO%PERMIT contains a set of bit flags describing access information about the file as follows:

<table>
<thead>
<tr>
<th>Bit Flag</th>
<th>Corresponding File</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE$ARCHIVE</td>
<td>Marked as having been copied to a backup device.</td>
</tr>
<tr>
<td>FILE$DIR</td>
<td>A subdirectory of the current directory. Each MS-DOS* directory contains two special files, &quot;.&quot; and &quot;.&quot;. These are directory aliases created by MS-DOS for use in relative directory notation. The first refers to the current directory, and the second refers to the current directory's parent directory.</td>
</tr>
<tr>
<td>FILE$HIDDEN</td>
<td>Hidden. It does not appear in the directory list you request from the command line, the Microsoft visual development environment browser, or File Manager.</td>
</tr>
<tr>
<td>FILE$READONLY</td>
<td>Write-protected. You can read the file, but you cannot make changes to it.</td>
</tr>
<tr>
<td>FILE$SYSTEM</td>
<td>Used by the operating system.</td>
</tr>
<tr>
<td>FILE$VOLUME</td>
<td>A logical volume, or partition, on a physical disk drive. This type of file appears only in the root directory of a physical device.</td>
</tr>
</tbody>
</table>
You can use the constant FILE$NORMAL to check that all bit flags are set to 0. If the derived-type element variable FILE$INFO%PERMIT is equal to FILE$NORMAL, the file has no special attributes. The variable FILE$INFO%NAME contains the short name of the file, not the full path of the file.

If an error occurs, call GETLASTERROR to retrieve the error message, such as:

- ERR$NOENT: The file or path specified was not found.
- ERR$NOMEM: Not enough memory is available to execute the command; or the available memory has been corrupted; or an invalid block exists, indicating that the process making the call was not allocated properly.

**Example**

```fortran
USE IFLPORT
CALL SHOWPERMISSION( )
END

SUBROUTINE SHOWPERMISSION()
! SUBROUTINE to demonstrate GETFILEINFO
!
USE IFLPORT
!
CHARACTER(80) files
INTEGER(4) handle, length
CHARACTER(5) permit
TYPE (FILE$INFO) info
!
WRITE (*, 900) ' Enter wildcard of files to view: '
900 FORMAT (A)
length = GETSTRQ(files)
handle = FILE$FIRST
DO WHILE (.TRUE.)
    length = GETFILEINFOQ(files, info, handle)
    IF ((handle .EQ. FILE$LAST) .OR. &
        (handle .EQ. FILE$ERROR)) THEN
```
SELECT CASE (GETLASTERRORQQ())
   CASE (ERR$NOMEM)
       WRITE (*,*) 'Out of memory'
   CASE (ERR$NOENT)
       EXIT
   CASE DEFAULT
       WRITE (*,*) 'Invalid file or path name'
END SELECT
END IF
permit = ' '
IF ((info%permit .AND. FILE$HIDDEN) .NE. 0) 
   permit(1:1) = 'H'
IF ((info%permit .AND. FILE$SYSTEM) .NE. 0) 
   permit(2:2) = 'S'
IF ((info%permit .AND. FILE$READONLY) .NE. 0) 
   permit(3:3) = 'R'
IF ((info%permit .AND. FILE$ARCHIVE) .NE. 0) 
   permit(4:4) = 'A'
IF ((info%permit .AND. FILE$DIR) .NE. 0) 
   permit(5:5) = 'D'
WRITE (*, 9000) info%name, info%length, permit
9000 FORMAT (1X, A20, I9, ' ',A6)
END DO
END SUBROUTINE

GETGID

Gets the group ID

Prototype

INTERFACE
   INTEGER FUNCTION GETGID ()
END FUNCTION GETGID
END INTERFACE

Description
This function returns an integer corresponding to the primary group of the user under whose identity this program is running.
On Win32 systems, this function returns the last subauthority of the security identifier for TokenPrimaryGroup for this process. This is unique on a local machine and unique within a domain for domain accounts.

NOTE. You should be aware that on Win32 systems, domain accounts and local accounts can overlap.

On Linux platforms, this function returns group identity for the current process.

Output
An integer representing the group that the currently logged in user belongs to.

GETLASTERROR
Gets the last error set

Prototype
INTERFACE
   INTEGER(4) FUNCTION GETLASTERROR ()
END FUNCTION GETLASTERROR
END INTERFACE
Description
This function returns the integer corresponding to the last runtime error value that was set.

For example, if you use an \texttt{ERR=} specifier on an I/O statement, your program will not abort in the event of an error. \texttt{GETLASTERROR} provides a way to determine what the error condition was, with a better degree of certainty than just examining \texttt{errno}. Your application code may then take appropriate action based upon the error number.

Output
Last error number into an integer.

\textbf{GETLASTERRORQQ}

\textit{Returns the last error set by a run-time procedure.}

Prototype
\begin{verbatim}
USE IFLPORT
or
INTERFACE
  INTEGER(4) FUNCTION GETLASTERRORQQ()
  END FUNCTION
END INTERFACE
\end{verbatim}

Usage
\begin{verbatim}
result = GETLASTERRORQQ()
\end{verbatim}

Results
The result is \texttt{INTEGER(4)}, and shows the most recent error code generated by a run-time procedure.
Descriptions that return a logical or integer value sometimes also provide an error code that identifies the cause of errors. `GETLASTERRORQQQ` retrieves the most recent error number, usually associated with `errno`. The error constants are in `IFLPORT.F90` The following table shows some of the portability library routines and the errors each routine produces:

<table>
<thead>
<tr>
<th>Runtime Routine</th>
<th>Runtime Routines Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNQQ</td>
<td>ERR$NOMEM, ERR$2BIG, ERR$INVAL, ERR$NOENT, ERR$NOEXEC</td>
</tr>
<tr>
<td>SYSTEMQQ</td>
<td>ERR$NOMEM, ERR$2BIG, ERR$NOENT, ERR$NOEXEC</td>
</tr>
<tr>
<td>GETDRIVESIZEQQ</td>
<td>ERR$INVAL, ERR$NOENT</td>
</tr>
<tr>
<td>GETDRIVESQQ</td>
<td>no error</td>
</tr>
<tr>
<td>GETDRIVEDIRQQ</td>
<td>ERR$NOMEM, ERR$RANGE</td>
</tr>
<tr>
<td>CHANGEDRIVEQQ</td>
<td>ERR$INVAL, ERR$NOENT</td>
</tr>
<tr>
<td>CHANGEDIRQQ</td>
<td>ERR$NOMEM, ERR$NOENT</td>
</tr>
<tr>
<td>MADEDIRQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$EXIST, ERR$NOENT</td>
</tr>
<tr>
<td>DELDIRQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$NOENT</td>
</tr>
<tr>
<td>FULLPATHQQ</td>
<td>ERR$NOMEM, ERR$INVAL</td>
</tr>
<tr>
<td>SPLITPATHQQ</td>
<td>ERR$NOMEM, ERR$INVAL</td>
</tr>
<tr>
<td>GETFILEINFOQQ</td>
<td>ERR$NOMEM, ERR$NOENT, ERR$INVAL</td>
</tr>
<tr>
<td>SETFILETIMEQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$INVAL, ERR$MFILE, ERR$NOENT</td>
</tr>
<tr>
<td>SETFILEACCESSQQ</td>
<td>ERR$NOMEM, ERR$INVAL, ERR$ACCES</td>
</tr>
<tr>
<td>DELFILESQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$NOENT, ERR$INVAL</td>
</tr>
<tr>
<td>RENAMEFILEQQ</td>
<td>ERR$NOMEM, ERR$ACCES, ERR$NOENT, ERR$XDEV</td>
</tr>
<tr>
<td>FINDFILEQQ</td>
<td>ERR$NOMEM, ERR$NOENT</td>
</tr>
<tr>
<td>PACKTIMEQQ</td>
<td>no error</td>
</tr>
<tr>
<td>UNPACKTIMEQQ</td>
<td>no error</td>
</tr>
<tr>
<td>COMMITQQ</td>
<td>ERR$BADF</td>
</tr>
<tr>
<td>GETCHARQQ</td>
<td>no error</td>
</tr>
</tbody>
</table>
GETLOG

Returns the user’s login name

Prototype

INTERFACE
SUBROUTINE GETLOG (NAME)
CHARACTER(LEN=*) NAME
END SUBROUTINE GETLOG
END INTERFACE

NAME the login name

Description

This function allows your application to determine the login name of the person running the application. The login name must be less than 64 characters. If the login name is longer than 64 characters, it will be truncated. The actual parameter corresponding to the formal parameter
NAME in the prototype should be long enough to hold the login name. If the supplied actual parameter is too short to hold the login name, the login name will be truncated.

**Output**

A character string in `NAME` corresponding to the login name of the person running the application. If the login name is shorter than the actual parameter corresponding to `NAME`, the login name is padded with blanks at the end, until it reaches the length of the actual parameter.

---

**GETPID**

*Gets the process ID*

**Prototype**

```fortran
INTERFACE
  INTEGER FUNCTION GETPID ()
  END FUNCTION GETPID
END INTERFACE
```

**Description**

This function returns the process ID of the current process.

**Output**

A unique integer corresponding to the system process identifier for the current process.
GETPOS

*Returns the current file position in bytes from the beginning of the file*

**Prototype**

```fortran
INTERFACE
  INTEGER FUNCTION GETPOS (LUNIT)
  INTEGER(4) LUNIT
END FUNCTION GETPOS
END INTERFACE
```

**Description**

Allows you to determine the current file position.

**Output**

An integer value representing the number of bytes from the beginning of the file. Equivalent to `FTELL`. Returns `EINVAL` in `errno` and a result of `-1` for an error.

---

GETSTATUSFPQQ

*Returns the floating-point processor status word.*

**Prototype**

```fortran
USE IFLPORT
```
or

INTERFACE
SUBROUTINE GETSTATUSFPQQ(STATUS)
INTEGER(2) STATUS
END SUBROUTINE
END INTERFACE

Usage
CALL GETSTATUSFPQQ (STATUS)
STATUS INTEGER (2). Floating-point co-processor status word.

The floating-point status word (FPSW) shows whether various floating-point exception conditions have occurred. After an exception occurs, the runtime system does not reset flags before performing additional floating-point operations. A status flag with a value of one thus shows there has been at least one occurrence of the corresponding exception. The following table lists the status flags and their values:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPSW$MSW_EM</td>
<td>Z'003F'</td>
<td>Status Mask (set all flags to 1)</td>
</tr>
<tr>
<td>FPSW$INVALID</td>
<td>Z'0001'</td>
<td>An invalid result occurred</td>
</tr>
<tr>
<td>FPSW$DENORMAL</td>
<td>Z'0002'</td>
<td>A denormals (very small number) occurred</td>
</tr>
<tr>
<td>FPSW$ZERODIVIDE</td>
<td>Z'0004'</td>
<td>A divide by zero occurred</td>
</tr>
<tr>
<td>FPSW$OVERFLOW</td>
<td>Z'0008'</td>
<td>An overflow occurred</td>
</tr>
<tr>
<td>FPSW$UNDERFLOW</td>
<td>Z'0010'</td>
<td>An underflow occurred</td>
</tr>
<tr>
<td>FPSW$INEXACT</td>
<td>Z'0020'</td>
<td>Inexact precision occurred</td>
</tr>
</tbody>
</table>

You can use a bit-wise AND on the status word returned by GETSTATUSFPQQ to determine which floating-point exception has occurred.

An exception is disabled if its flag is set to 1 and enabled if its flag is cleared to 0. By default, the denormal, underflow and inexact precision exceptions are disabled, and the invalid, overflow and divide-by-zero exceptions are enabled. Exceptions can be enabled and disabled by clearing
and setting the flags with \texttt{SETCONTROLFPQQ}. You can use \texttt{GETCONTROLFPQQ} to determine which exceptions are currently enabled and disabled.

If an exception is disabled, it does not cause an interrupt when it occurs. Instead, floating-point processes generate an appropriate special value (\texttt{NaN} or signed infinity), but the program continues. When printed, the runtime-system represents a \texttt{NaN} as \texttt{?????}, positive infinity as \texttt{+++++}, and negative infinity as \texttt{--------}. You can find out which exceptions (if any) occurred by calling \texttt{GETSTATUSFPQQ}.

If errors on floating-point exceptions are enabled (by clearing the flags to 0 with \texttt{SETCONTROLFPQQ}), an interrupt is generated when the exception occurs. By default, these interrupts cause run-time errors, but you can capture the interrupts with \texttt{SIGNALQQ} and branch to your own error-handling routines.

\textbf{Example}

\begin{verbatim}
! Program to demonstrate GETSTATUSFPQQ
USE IFLPORT
INTEGER(2) status
CALL GETSTATUSFPQQ(status)
! check for divide by zero
IF (IAND(status, FPSW$ZERODIVIDE) .NE. 0) THEN
  WRITE (*,*) 'Divide by zero occurred.', &
             'Look for NaN or signed infinity in resultant data.'
ELSE
  PRINT *, 'Divide by zero flag was not set'
END IF
END
\end{verbatim}
GETSTRQQ

Reads a character string from the keyboard using buffered input.

Prototype

USE IFLPORT
or
INTERFACE
  INTEGER(4) FUNCTION GETSTRQQ(BUFFER)
   CHARACTER(LEN=*) BUFFER
  END FUNCTION
END INTERFACE

Usage

result = GETSTRQQ (BUFFER)
BUFFER CHARACTER(LEN=*). Character value returned from keyboard, padded on the right with blanks.

Results

The result is the number of characters placed in BUFFER. The function continues, until the user presses Return or Enter.

Example

! Program to demonstrate GETSTRQQ
USE IFLPORT
INTEGER(4) length, result
CHARACTER(80) prog, args
WRITE (*, '(A,)') ' Enter program to run: '
length = GETSTRQQ (prog)
WRITE (*, '(A,)') ' Enter arguments: '
length = GETSTRQQ (args)
result = RUNQQ (prog, args)
IF (result .EQ. -1) THEN
  WRITE (*,*) 'Couldn''t run program'
ELSE
  WRITE (*, '(A, Z4, A)') 'Return code : ', result, 'h'
END IF
END

GETTIM

*Returns the time*

**Prototype**

INTERFACE

SUBROUTINE GETTIM(HOUR,MIN,SEC,HHTS)
  INTEGER(4), INTENT(OUT) :: HOUR,MIN,SEC,HHTS
END SUBROUTINE

SUBROUTINE GETTIM_DVF(HOUR,MIN,SEC,HHTS)
  INTEGER(2), INTENT(OUT) :: HOUR,MIN,SEC,HHTS
END SUBROUTINE

END INTERFACE

HOUR current system hour
MINUTE current system minutes
SECOND current system seconds
HUNDREDTH current system hundredths of a second

**Description**

This function gets the current system time in hours, minutes, seconds, and hundredths of a second. The time units are returned as separate integers to the calling routine.
Output
The time units are returned as separate integers to the calling routine.

GETTIMEOFDAY

Returns second and milliseconds since 00:00 Jan 1, 1970.

Prototype

```
INTERFACE
  SUBROUTINE GETTIMEOFDAY(RET, ERR)
      INTEGER(4) RET(2), ERR
  END SUBROUTINE
END INTERFACE
```

```
RET(1) Output. INTEGER(4). Returned seconds.
RET(2) Output. INTEGER(4). Returned milli- or microseconds.
```

Description
This function places the number of seconds and milli- or microseconds since 00:00 Jan 1, 1970 and places seconds in array element RET(1) and milliseconds in array element RET(2).

For Linux*, a system routine gettimeofday() is used.

For Windows*, a system routine _ftime() is used. Note that _ftime() returns only milliseconds.

Output
If error occurs, ERR contains a value equal to -1, and array RET contains zeros.
GETUID

*Gets the user ID of the calling process*

**Prototype**

```
INTERFACE
  INTEGER FUNCTION GETUID()
  END FUNCTION GETUID
END INTERFACE
```

**Description**

This function returns an integer corresponding to the user identity under which this program is running. This function returns the last subauthority of the security identifier for this process. This is unique on a local machine and unique within a domain for domain accounts.

**NOTE.** You should be aware that on Win32 systems, domain accounts and local accounts can overlap.

**On Linux platforms,** this function returns user identity for the current process.

**Output**

The user **ID** of the calling process.
**GMTIME**

*Converts the given elapsed time to the current system time*

---

**Prototype**

**IA-32 systems**

SUBROUTINE GMTIME (STIME, DATEARRAY)
   INTEGER (4) STIME
   INTEGER (4) DATEARRAY (9)
END SUBROUTINE

**Itanium®-based systems**

SUBROUTINE GMTIME (STIME, DATEARRAY)
   INTEGER (8) STIME
   INTEGER (4) DATEARRAY (9)
END SUBROUTINE

---

**STIME**

STIME represents an elapsed time in seconds since midnight, January 1, 1970, in GMT. You can obtain this value from GETTIMEOFDAY, if you adjust the result of GETTIMEOFDAY for your local time zone.

**DATEARRAY (1 : 9)** contains the current date and system time, GMT.

- DATEARRAY (1) seconds (0-59)
- DATEARRAY (2) minutes (0-59)
- DATEARRAY (3) hours (0-23)
- DATEARRAY (4) day of month (1-31)
- DATEARRAY (5) month (0-11)
- DATEARRAY (6) year in century (0-99)
- DATEARRAY (7) day of week (0-6, 0 is Sunday)
- DATEARRAY (8) day of year (0-365)
- DATEARRAY (9) daylight savings (1, if in effect; else 0)
Description
This function converts a given elapsed time in seconds into the current system date, GMT.

NOTE. This function may give problems with the year 2000. Use DATE_AND_TIME instead.

Output
The current date and system time for Greenwich Mean Time, in DATEARRAY.

HOSTNAM
Retrieves the current host computer name.

Prototype
USE IFLPORT
or
INTERFACE
   INTEGER(4) FUNCTION HOSTNM(NAME)
   CHARACTER(LEN=*) , INTENT(OUT) :: NAME
END FUNCTION
END INTERFACE

Description
Retrieves the current host computer name. This function is exactly the same as HOSTNM.
Usage
result = HOSTNAM (NAME)
NAME CHARACTER(LEN=*)

Thename of the current computer host is returned in NAME. The buffer provided should be at least as long as MAX_COMPUTERNAME_LENGTH, which is defined in the IFLPORT module.

Results
The result is zero if successful. If NAME is not long enough to contain all of the host name, the function truncates the host name and returns -1.

Example
USE IFLPORT
CHARACTER(LEN=15) HOSTNAME
INTEGER(4) ISTATUS
ISTAT = HOSTNAM (HOSTNAME)
PRINT *, HOSTNAME, ' ISTATUS = ', ISTATUS
END

HOSTNM

Gets the current host name

Prototype
INTERFACE
  SUBROUTINE HOSTNM( NAME)
  CHARACTER(LEN=*) NAME
  END SUBROUTINE HOSTNM
END INTERFACE
NAME

A CHARACTER variable or array element large enough to hold the name of the current host computer. On Win32 systems, the maximum host name length is 15.

Description

This function retrieves the current host computer name.

Output

An ASCII character string specifying the name of the host computer where your application is executing.

IARG

*Returns the number of arguments on the command line*

Prototype

INTERFACE
  INTEGER(4) FUNCTION IARG ()
  END FUNCTION
END INTERFACE

Description

This function returns the number of arguments on the command line, not including the command itself. Synonym of IARGC.

Output

An integer number of arguments. If no arguments are passed to the program, IARGC returns zero. Otherwise IARG returns a count of the arguments that follow the program name on the command line.
Example
The statement PRINT *, IARG() prints a count of the arguments passed to the program.

IARGC

Returns the number of arguments on the command line

Prototype
INTERFACE
   INTEGER(4) FUNCTION IARGC ()
   END FUNCTION IARC
END INTERFACE

Description
This function returns the number of arguments on the command line, not including the command itself. Synonym of IARG.

Output
An integer number of arguments. If no arguments are passed to the program, IARGC returns zero. Otherwise IARGC returns a count of the arguments that follow the program name on the command line.

Example
The statement PRINT *, IARGC() prints a count of the arguments passed to the program.
IDATE

*Returns the current system date*

**Prototype**

```plaintext
INTERFACE
    SUBROUTINE IDATE (MONTH, DAY, YEAR)
        INTEGER (4) INTENT (OUT) :: MONTH, DAY, YEAR
    END SUBROUTINE IDATE
    SUBROUTINE F_IDATE (SDATE)
        INTEGER (4) INTENT (OUT) :: SDATE (3)
    END SUBROUTINE F_IDATE
END INTERFACE
```

**MONTH**
Output. Current system month

**DAY**
Output. Current system day

**YEAR**
Output. Current system year as an offset from 1900

**SDATE**
Output. Three-element array that holds day as element 1, month as element 2, and year as element 3. The month is between 1 and 12 and the year is greater than or equal to 1969.

**Description**

This routine returns the current system month, day, and year.

**Example**

The statement `CALL IDATE (MONTH, DAY, YR)` sets `MON` to the month number, `DAY` to the day of the month, and `YR` to the two-digit year representation (for example, 69 for the year 1969).
IDATE4

Returns the current system date

Prototype

INTERFACE

SUBROUTINE IDATE4(MONTH,DAY,YEAR)
  INTEGER(4), INTENT(OUT) :: MONTH, DAY, YEAR
END SUBROUTINE

SUBROUTINE F_IDATE4(SDATE)
  INTEGER(4), INTENT(OUT) :: SDATE(3)
END SUBROUTINE

END INTERFACE

DATEARRAY an integer array.

Description

This routine returns the current system month, day, and year.
This function is year-2000 compliant.

Output

DATEARRAY (1) Current system day.
DATEARRAY (2) Current system month.

NOTE. This subroutine is not year-2000 compliant. Use DATE_AND_TIME or IDATE4 instead.
**DATEARRAY (3)**

Current system year as an offset from 1900, if the year is less than 2000. For years greater than or equal to 2000, this element simply returns the integer year, such as 2003.

---

**IDFLOAT**

*Converts INTEGER (4) argument to DOUBLE PRECISION.*

**Prototype**

```fortran
INTERFACE
  REAL(8) FUNCTION IDFLOAT (IARG)
    !MSSATTRIBUTES ALIAS:'idfloat_':'IDFLOAT
    INTEGER(4), INTENT(IN) :: IARG
  END FUNCTION
END INTERFACE
```

**Input**

IARG

Input. INTEGER(4). Integer argument to be converted to DOUBLE PRECISION.

**Output**

The result is of REAL(8), a DOUBLE PRECISION.
IEEE_FLAGS

Sets or gets IEEE* flags for rounding direction and precision as well as queries or clears exception status.

Prototype

INTERFACE
   INTEGER FUNCTION IEEE_FLAGS (ACTION, MODE, IN, OUT)
   CHARACTER(LEN=*) ACTION, MODE, IN, OUT
END FUNCTION IEEE_FLAGS
END INTERFACE

ACTION  Input  CHARACTER(LEN=*) , literal. The values are one of the following names of the procedures to set the flags for: 'GET', 'SET', 'CLEAR', or 'CLEARALL'.

MODE    Input  CHARACTER(LEN=*) , literal. The values are: 'direction', 'precision', 'exception'

IN      Input CHARACTER(LEN=*) , literal. The flag values are: 'inexact', 'underflow', 'overflow', 'division', 'invalid', 'all', 'common', 'nearest', 'tozero', 'negative', 'positive', 'extended', 'double', 'single' or '1'.

OUT     Output. Must be at least CHARACTER*9 , literal. One of the values listed for the IN parameter.

The meanings of the values for in and out are summarized in the following table.

<table>
<thead>
<tr>
<th>Values of in and out</th>
<th>Indicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>'nearest'(default), 'tozero', 'negative', 'positive'</td>
<td>Rounding direction flags</td>
</tr>
<tr>
<td>'single', 'double' (default for Windows), 'extended' (default for Linux)</td>
<td>Rounding precision flags</td>
</tr>
</tbody>
</table>
Portability Functions

Values of in and out | Indicate
---|---
'inexact', 'underflow', 'overflow', 'division', 'invalid' | Exception flags
'all' | All five exception flags (above)
'common (exceptions: 'invalid', 'division', 'overflow')' | Three exceptions

1 Exceptions are listed in the order from the highest to the lowest priority.
2 The Intel Fortran Compiler supports an additional floating-point exception, denormal. For details, see Intel® Fortran Compiler User’s Guide.

Description

IEEE_FLAGS is an integer-valued function that gets, sets or clears IEEE flags. The function enables you to control rounding direction and rounding precision, queries exception status or clears exception status. The flags information is returned as a set of 1-bit flags. The actions (GET, SET, CLEAR, and CLEARALL) function in a different way with mode equal to 'direction' or 'precision' and mode equal to 'exception'.

GET:

- Returns the direction or precision mode information:
  
  ```fortran
  ieee_flags ('get', 'precision', '', out)
  ```

- Gets the designated bits for the exception flags status:
  
  ```fortran
  ieee_flags ('get', 'exception', 'division', out)
  ```

SET:

- Sets the direction and precision modes to one of their values:
  
  ```fortran
  ieee_flags ('set', 'precision', 'single', out)  
  ```
  
  (the last argument can be '', a null string)

  ```fortran
  ieee_flags ('set', 'direction', 'tozero', '')
  ```

- Sets the exception flags (sets their designated bits in the FPU status word to 1):
  
  ```fortran
  ieee_flags ('set', 'exception', 'overflow', '')
  ```
CLEAR:
- Sets the values of direction or precision to their defaults: 
  ```fortran
  ieee_flags('clear', 'direction', '', '')
  (sets 'direction' = 'nearest')
  ieee_flags('clear', 'precision', '', '')
  (sets 'precision' = 'double' (Windows))
  ```
- Clears the exception flags (sets their corresponding bits in the FPU status word to 0):
  ```fortran
  iflag=ieee_flags('clear', 'exception', & 'overflow', '')
  ```

CLEARALL:
- Restores default direction and precision.
- Sets all exception flags to 0.
  ```fortran
  ieee_flags('clearall', '', '', '')
  ```
The values for in and out depend on the action and mode they are used with. The interaction of the parameters is summarized in the table that follows.

<table>
<thead>
<tr>
<th>Parameter Values</th>
<th>Functionality and Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>action</strong></td>
<td><strong>mode</strong></td>
</tr>
<tr>
<td>GET</td>
<td>direction</td>
</tr>
<tr>
<td></td>
<td>precision</td>
</tr>
<tr>
<td></td>
<td>exception</td>
</tr>
</tbody>
</table>

continued
Portability Functions

Example 1. Determine what is the highest priority exception that has a flag raised. Pass the input argument \( in \) as the null string:

```plaintext
INTEGER*4 iflag
CHARACTER*9 out
iflag = ieee_flags('get', 'exception', '', out)
PRINT *, out, ' flag raised'
```
Example 2. Set rounding direction to round toward zero.
INTEGER*4 iflag
CHARACTER*1 mode, out, in
iflag = ieee_flags('set', 'direction', 'tozero', out)

Example 3. Clear rounding direction to default (nearest):
INTEGER*4 iflag
CHARACTER*1 out, in
iflag = ieee_flags('clear', 'direction', '', '')

IEEE_HANDLER
Establishes a handler for IEEE exceptions.

Prototype
IA-32 systems
INTERFACE
INTEGER(4) FUNCTION IEEE_HANDLER (ARG_ACTION,&
ARG_EXCEPTION,HANDLER)
CHARACTER(LEN=*) ARG_ACTION, ARG_EXCEPTION
INTERFACE
SUBROUTINE HANDLER (SIGNO,SIGINFO)
INTEGER(4), INTENT(IN)::SIGNO, SIGINFO
END SUBROUTINE
END INTERFACE
END FUNCTION IEEE_HANDLER
END INTERFACE
**Itanium®-based systems**

INTERFACE

INTEGER(8) FUNCTION IEEE_HANDLER (ARG_ACTION,&
ARG_EXCEPT,HANDLER)

CHARACTER(LEN=*) ARG_ACTION, ARG_EXCEPT

INTERFACE

SUBROUTINE HANDLER (SIGNO, SIGINFO)

INTEGER(4), INTENT(IN)::SIGNO, SIGINFO

END SUBROUTINE

END INTERFACE

END FUNCTION IEEE_HANDLER

END INTERFACE

**Description**

IEEE_HANDLER calls HANDLER subroutine to establish a handler for IEEE exceptions.

**Output**

Returns 0 if executes successfully, 1 otherwise.

---

**IERRNO**

*Returns the last error code generated*

**Prototype**

INTERFACE

INTEGER(4) FUNCTION IERRNO()

END FUNCTION IERRNO

END INTERFACE
Description
This function returns the number of the last detected error from any module that returns error codes.

Output
The last error code generated.

IFL_RUNTIME_INIT
Initializes the Fortran runtime system, complete with command line arguments

Prototype
INTERFACE
SUBROUTINE IFL_RUNTIME_INIT ( NCMDARGS, ARGS )
INTEGER(4) NCMDARGS
INTEGER(1), POINTER :: ARGS
END INTERFACE

NCMDARGS The number of arguments on the command line for the application.
ARGS A pointer to an array of character strings, where each string makes up a command line argument

Description
This routine initializes the Fortran runtime system, including supplying command line arguments. Normally this is not required. However, if your main program is written in C, or your main program is a WINMAIN in C or C++ for Windows®, there may be circumstances where you wish to set the command line arguments for the Fortran portion of the program, and perform other initialization. For IA-32 Windows applications, you would call this routine from C as:
extern void IFL_RUNTIME_INIT ( int *argc, char *
argv[]);

int main(int *argc, char *argv[]) 
{
    IFL_RUNTIME_INIT(argc, argv);
}

**IFLOAT**
*Converts an input value to a real value*

**Prototype**

```plaintext
INTERFACE
    REAL(4) FUNCTION IFLOAT (IN)
    INTEGER(2) IN
    END FUNCTION IFLOAT
END INTERFACE
```

**Description**

*Converts an input value to REAL(4).*

**Output**

*The equivalent REAL(4) value.*
IFLOATI

Converts an INTEGER(2) to a REAL type

Prototype

INTERFACE
  REAL(4) FUNCTION IFLOATI (INPUT)
  INTEGER(2), INTENT(IN)::INPUT
END FUNCTION IFLOATI
END INTERFACE

INPUT
a scalar INTEGER (KIND=2) value

Description

IFLOATI is an elemental function that converts a scalar integer (KIND=2) type to REAL(4).

Output
The integer value converted to REAL.

IFLOATJ

Converts an integer to a real value

Prototype

INTERFACE
  REAL(4) FUNCTION IFLOATJ (IN)
  INTEGER(4) IN
END FUNCTION IFLOATJ
END INTERFACE
IN an INTEGER(4) expression

**Description**
Converts an input value to REAL(4).

**Output**
The equivalent REAL(4) value.

---

**IMOD**

*Returns the remainder of division of the first argument by the second argument*

**Prototype**

```
INTERFACE
  INTEGER(2) FUNCTION IMOD(A, P)
  !MS$ATTRIBUTES ALIAS:'imod_'::IMOD
  INTEGER(2), INTENT(IN) :: A, P
END FUNCTION
END INTERFACE
```

**Syntax**

```
result = IMOD (A, P)
```

**A**
Input. Must be of type INTEGER(2).

**ARG2**
Input. Must have the same type and kind parameters as A.

**Results**
The result type is INTEGER(2). If P is not equal to zero, the value of the result is:

```
result = A - INT(A/P) * P.
```

If P is equal to zero, the result is undefined.
INMAX

*Returns the maximum positive integer*

**Prototype**

```
INTERFACE
  INTEGER(4) FUNCTION INMAX (INPUT)
  INTEGER(4), INTENT(IN)::INPUT
END FUNCTION INMAX
END INTERFACE
```

**Description**

This function returns the maximum positive value for an `INTEGER(4)`.

**Output**

The maximum 4-byte-signed integer.

---

INTC

*Converts INTEGER(4) to INTEGER(2)*

**Prototype**

```
INTERFACE
  INTEGER(2) FUNCTION INTC (IN)
  INTEGER IN
END FUNCTION INTC
END INTERFACE
```

**Description**

This function converts a value of type `INTEGER(4)` to `INTEGER(2)`.

**Input**

Any `INTEGER(4)` value or expression.
Description
Converts an INTEGER(4) value or expression to an INTEGER(2) value.

Output
The value of IN converted to a type INTEGER(2). Overflow is ignored.

IRAND
Generates pseudorandom numbers.

INTERFACE
   INTEGER(4) FUNCTION IRAND()
   INTEGER(4) ISEED
END FUNCTION IRAND
END INTERFACE

Description
Generates pseudorandom numbers.

Class
Elemental nonstandard function.

Result Type and Type Parameter
INTEGER(4) type.

Result Value
IRAND generates numbers in the range 0 through $2^{31}$.

Example
INTEGER(4) rn
rn = IRAND()
IRANDM

A synonym for IRAND.

Description
This function is a synonym for IRAND. It takes the same parameters and produces the same result.

IRANGET

Gets a random number

Prototype

```fortran
INTERFACE
  SUBROUTINE IRANGET (S)
    INTEGER(4) S
  END SUBROUTINE IRANGET
END INTERFACE
```

S an integer expression

Description
Returns the next in a series of pseudo-random numbers.

NOTE. For details about restarting the pseudorandom number generator used by IRAND and RAND, see the SRAND section.
Output
An integer pseudo random number.

IRANSET
Sets the seed for a sequence of pseudo-random numbers

Prototype
INTERFACE
   SUBROUTINE IRANSET (ISEED)
   INTEGER(4) ISEED
   END SUBROUTINE IRANSET
END INTERFACE

ISEED the new seed for a sequence of pseudo-random numbers

Description
Sets the seed for a sequence of random numbers.

Output
Changes the internal seed. Not thread-safe.

ISATTY
Returns true if the specified unit number is a terminal

Prototype
INTERFACE
**ISATTY**

*Description*

This function returns true if the specified unit number is a terminal. The unit must be connected. LUN must be an integer expression. If LUN is out of range, zero is returned. LUN must map to a connected Fortran logical unit at the time of the call. If LUN corresponds to a unit that is not connected, zero is returned.

*Output*

*.TRUE.* for a logical unit connected to a terminal device. *.FALSE.* otherwise.

**ITIME**

*Prototype*

```fortran
INTERFACE
   SUBROUTINE ITIME(TIME_ARRAY)
   INTEGER(4) TIME_ARRAY(3)
END FUNCTION ITIME
END INTERFACE
```

*Description*

This function returns the time in numeric form in a 3-element array.
Output


---

**JABS**

*Returns the absolute value*

**Prototype**

```fortran
INTERFACE
  INTEGER FUNCTION JABS (I)
  INTEGER(4), INTENT(IN):: I
  END FUNCTION JABS
END INTERFACE
```

---

**JDATE**

*Returns the date in ASCII*

**Prototype**

```fortran
INTERFACE
  FUNCTION JDATE()
  CHARACTER(LEN=8):: JDATE
  END FUNCTION JDATE
END INTERFACE
```
Description
This function returns an 8-character ASCII string with the Julian date, (day of the year).

Output
An 8-character ASCII string with the Julian date in the form yyddd.

NOTE. Use of this function is discouraged due to possible problems with the change to the year 2000. Use DATEANDTIME, a standard function instead.

JDATE4
Returns the date in ASCII

Prototype
INTERFACE
    SUBROUTINE JDATE4(CURRENTDATE)
    CHARACTER(LEN=10) CURRENTDATE
    END SUBROUTINE JDATE4
END INTERFACE

Description
This function returns an 10-character ASCII string with the Julian date, (day of the year).

Output
An 10-character ASCII string with the Julian date in the form yyyyddd.
**KILL**

_Sends a signal to a process given by ID._

**Prototype**

```fortran
USE IFLPORT
or
INTERFACE
  INTEGER(4) FUNCTION KILL (PID, SIGNUM)
  INTEGER(4) PID, SIGNUM
END FUNCTION
END INTERFACE
```

- **PID**  
  ID of a process to be signaled

- **SIGNUM**  
  Signal value

**Description**

This function sends a signal `SIGNUM` to the process specified by `PID`. This function requires that the program executing this function have `TERMINATE_PROCESS` access to the process being killed.

**Output**

If successful, zero.
LCWRQQ

Sets the value of the floating-point processor control word.

Prototype

USE IFLPORT
or
INTERFACE
  SUBROUTINE LCWRQQ(CONTROL)
    INTEGER(2) CONTROL
  END SUBROUTINE
END INTERFACE

Usage

CALL LCWRQQ (CONTROL)
CONTROL INTEGER(2). Floating-point processor control word.
LCWRQQ performs the same function as SETCONTROLFPQQ and is provided for compatibility.

Example

USE IFLPORT
INTEGER(2) control
CALL SCWRQQ(control) ! get control word
! Set control word to make processor round up
control = control .AND. (.NOT. FPCW$MCW_RC) ! Clear ! control
  ! word with inverse
  ! of
  ! rounding control mask
control = control .OR. FPCW$UP ! Set control word ! to round up
CALL LCWRQQ(control)
FUNCTION
9000 FORMAT (1X, A, Z4)
END

**LEADZ**

*Returns the number of leading zero bits in an integer.*

**Prototype**

```fortran
INTERFACE
   INTEGER(4) FUNCTION LEADZ(INPUT)
   !MS$ATTRIBUTES ALIAS:'leadz_'::LEADZ
   INTEGER(4), INTENT(IN) :: INPUT
END FUNCTION
END INTERFACE
```

**Syntax**

```fortran
result = LEADZ (I)
```

*I*  
Output. INTEGER. The number of leading zero bits.

**Results**

The result type is the same as I. The result value is the number of leading zeros in the binary representation of the integer I.

**Example**

```fortran
INTEGER(8) INT, TWO
PARAMETER (TWO=2)
DO INT = -1, 40
   TYPE *, LEADZ(TWO**INT) !Prints 64 down to 23
   ! leading zeros
ENDDO
```
**LNBLNK**

Locates the position of the last nonblank character in a string.

**Prototype**

```fortran
INTERFACE
 INTEGER(4) FUNCTION LNBLNK(STRING)
   !MS$ATTRIBUTES ALIAS:'lnblnk_'::LNBLNK
   CHARACTER(LEN=*) , INTENT(IN) :: STRING
 END FUNCTION
END INTERFACE
```

**Syntax**

```fortran
result = LNBLNK (string)
string       Character(len=*). String to be searched.
             Cannot be an array.
```

**Results**

The result is of type INTEGER(4). The result is the index of the last nonblank character in string.

**Example**

```fortran
USE IFLPORT
INTEGER(4) POS
POS = LNBLNK (' HELLO WORLD ') ! Returns 12
POS = LNBLNK (' ')  ! Returns 0
```
LONG

Returns an INTEGER(2) value as an INTEGER(4) type

Prototype
USE IFLPORT

Usage
result = LONG (INT2)
INT2 Input. INTEGER(2). Value to be converted.

Results
The result is of type INTEGER(4). The result is the value of INT2 with the type of INTEGER(4). The upper 16 bits of the result are zeros and the lower 16 are equal to INT2.

LSTAT

Return detailed information about a file.

Prototype
USE IFLPORT

Usage
result = LSTAT (NAME, STATH)
NAME Input. CHARACTER(LEN=*). Name of the file to examine.
STATH Output. INTEGER(4). One-dimensional array with the size of 12. See STAT for the possible values returned in STATH.
Results

The result is of type INTEGER(4). The result is zero if successful; otherwise an error code, see IERRNO.

LSTAT returns detailed information about the file named in NAME. Currently, LSTAT returns exactly the same information as STAT because there are no symbolic links. STAT is the referred function.

Example

USE IFLPORT
INTEGER(4)INFO_ARRAY(12), ISTATUS
CHARACTER(LEN=20)FILENAME
PRINT*, 'Enter name of file to examine: '
READ*, FILENAME
ISTATUS = LSTAT(FILENAME, INFO_ARRAY)
IF (.NOT. ISTATUS) THEN
  PRINT*, INFO_ARRAY
ELSE
  PRINT*, 'Error ', ISTATUS
ENDIF

LTIME

Returns the components of the local time zone time

Prototype

IA-32 systems

INTERFACE
  SUBROUTINE LTIME(TIME, ARRAY)
  INTEGER(4) TIME, ARRAY(9)
  END SUBROUTINE
END INTERFACE
Itanium®-based systems:

INTERFACE
  SUBROUTINE LTIME(TIME, ARRAY)
  INTEGER(8) TIME
  INTEGER(40) ARRAY(9)
  END SUBROUTINE
END INTERFACE

INTEGER(4) TIME  An elapsed time in seconds since 00:00:00
Greenwich Mean Time, January 1, 1970

Description

Returns the components of the local time zone time in a
nine-element array.

ARRAY INTEGER (4) one-dimensional array with nine
  elements to contain local date and time data
derived from TIME returned as follows:

  ARRAY (1)  Seconds (0-59)
  ARRAY (2)  Minutes (0-59)
  ARRAY (3)  Hours (0-23)
  ARRAY (4)  Day of month (1-31)
  ARRAY (5)  Month (0-11)
  ARRAY (6)  Year number in century (0-99)
  ARRAY (7)  Day of week (0-6, where 0 is Sunday)
  ARRAY (8)  Day of year (1-365)
  ARRAY (9)  1 if daylight savings time is in effect; otherwise, 0.

NOTE. This function is not year 2000 compliant, use DATE_AND_TIME
instead.
MAKEDIRQQ

Creates a new directory with a specified name.

Prototype

USE IFLPORT
or
INTERFACE
  LOGICAL(4) FUNCTION MAKEDIRQQ(DIRNAME)
  CHARACTER(*) DIRNAME
  END FUNCTION
END INTERFACE

Usage

result = MAKEDIRQQ (DIRNAME)
DIRNAME CHARACTER(LEN=*). Name and path of the directory you want created.

Results

MAKEDIRQQ returns .TRUE. if successful; otherwise, .FALSE..
MAKEDIRQQ can create only one directory at a time. You cannot create a new directory and a subdirectory below it in a single command.
MAKEDIRQQ does not translate path delimiters. You can use either slash (/) or backslash () as valid delimiters.
If an error occurs, you should call GETLASTERRORQQ to determine the problem. Possible errors include:

ERR$ACCES  Permission denied. The file's (or directory's) permission setting does not allow the specified access.
ERR$EXIST  The directory already exists.
ERR$NOENT  The file or path specified was not found.
Example

USE IFLPORT
LOGICAL(4) result
result = MAKEDIRQQ('mynewdir')
IF (result) THEN
  WRITE (*,*) 'New subdirectory successfully created'
ELSE
  WRITE (*,*) 'Failed to create subdirectory'
END IF
END

MATHERRQQ

Handles run-time math errors.

Prototype

INTERFACE
  SUBROUTINE MATHERRQQ( NAME,NLEN,INFO,RETCODE)
    CHARACTER(LEN=*) NAME
    INTEGER(2) :: NLEN, RETCODE
    STRUCTURE /MTH$E_INFO/
      INTEGER*4 ERRCODE ! INPUT : One of the MTH$ values above
      INTEGER*4 FTYPE ! INPUT : One of the TY$ values above
    UNION
      MAP
        REAL*4 R4ARG1 ! INPUT : First argument
        CHARACTER*12 R4FILL1
        REAL*4 R4ARG2 ! INPUT : Second argument
        !(if any)
  END INTERFACE
CHARACTER*12 R4FILL2
REAL*4 R4RES ! OUTPUT : Desired result
CHARACTER*12 R4FILL3
END MAP
MAP
REAL*8 R8ARG1 ! INPUT : First argument
CHARACTER*8 R8FILL1
REAL*8 R8ARG2 ! INPUT : Second argument
! (if any)
CHARACTER*8 R8FILL2
REAL*8 R8RES ! OUTPUT : Desired result
CHARACTER*8 R8FILL3
END MAP
MAP
COMPLEX*8 C8ARG1 ! INPUT : First argument
CHARACTER*8 C8FILL1
COMPLEX*8 C8ARG2 ! INPUT : Second argument
! (if any)
CHARACTER*8 C8FILL2
COMPLEX*8 C8RES ! OUTPUT : Desired result
CHARACTER*8 C8FILL3
END MAP
MAP
COMPLEX*16 C16ARG1 ! INPUT : First argument
COMPLEX*16 C16ARG2 ! INPUT : Second argument
! (if any)
COMPLEX*16 C16RES ! OUTPUT : Desired result
END MAP
END UNION
END STRUCTURE
RECORD /MTH$E_INFO/ info
END SUBROUTINE
END INTERFACE

NOTE. Due to the fact that you cannot use old VAX* style structures in Fortran 95 modules with Intel Fortran, there is no interface for this function in IFLPORT.F90

Usage

CALL MATHERRQ (NAME, NLEN, INFO, RETCODE)

NAME CHARACTER(LEN=*) . On return, this variable contains the name of the function causing the error. The parameter NAME is a typeless version of the function called. For example, if an error occurs in a SIN function, the name will be returned as SIN for real arguments and CSIN for complex arguments even though the function may have actually been called with an alternate name such as DSIN or CDSIN, or with SIN and complex arguments.

NLEN INTEGER(2) . Number of characters returned in NAME.

INFO UNION containing data about the error. The MTH$E_INFO union is defined above.

RETCODE INTEGER(2) . Return code passed back to the run-time library. The value of RETCODE should be set by the user's MATHERRQ routine to indicate whether the error was resolved. Set this value to 0 to indicate that the error was not resolved and that the program should fail with a run-time error. Set it to any nonzero value to indicate that the error was resolved and the program should continue.

The ERRCODE element in the MTH$E_INFO structure specifies the type of math error that occurred, and can have one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$E_DOMAIN</td>
<td>Argument domain error</td>
</tr>
</tbody>
</table>
The `FTYPE` element of the `INFO` structure identifies the data type of the math function as `TY$REAL4`, `TY$REAL8`, `TY$CMPLX4`, or `TY$CMPLX8`. Internally, `REAL(4)` and `COMPLEX(4)` arguments are converted to `REAL(8)` and `COMPLEX(8)`. In general, a `MATHERRQQ` function should test the `FTYPE` value and take separate action for `TY$REAL8` or `TY$CMPLX8` using the appropriate mapped values. If you want to resolve the error, set the `R8RES` or `C8RES` field to an appropriate value such as 0.0. You can do calculations within the `MATHERRQQ` function using the appropriate `ARG1` and `ARG2` fields, but avoid doing any calculations that would cause an error resulting in another call to `MATHERRQQ`.

**NOTE.** You cannot use `MATHERRQQ` in DLLs or in a program that links with a DLL.

### NARGS

*Returns the number of arguments on the command line*

**Prototype**

```fortran
INTERFACE
  INTEGER(4) FUNCTION NARGS ()
END INTERFACE
```
END FUNCTION NARGS
END INTERFACE

**Description**
This function returns the number of arguments on the command line, not including the invoking command itself.

**Output**
An integer value, zero or positive, indicating the number of arguments on the command line invoking your program.

---

**NUMARG**
*Returns the number of arguments on the command line*

**Prototype**

INTERFACE
  INTEGER(4) FUNCTION NUMARG ()
  END FUNCTION NUMARG
END INTERFACE

**Description**
This function returns the number of arguments on the command line, not including the invoking command itself.

**Output**
An integer value, zero or positive, indicating the number of arguments on the command line invoking your program.
PACKTIMEQQ

Packs time and date values.

**Prototype**

```fortran
USE IFLPORT
or

**IA-32 systems**

INTERFACE
  SUBROUTINE PACKTIMEQQ(TIMEDATE, IYR, IMON, IDAY, IHR, IMIN, ISEC)
    INTEGER(4) TIMEDATE
    INTEGER(2) IYR, IMON, IDAY, IHR, IMIN, ISEC
  END SUBROUTINE
END INTERFACE

**Itanium®-based systems:**

INTERFACE
  SUBROUTINE PACKTIMEQQ(TIMEDATE, IYR, IMON, IDAY, IHR, IMIN, ISEC)
    INTEGER(8) TIMEDATE
    INTEGER(2) IYR, IMON, IDAY, IHR, IMIN, ISEC
  END SUBROUTINE
END INTERFACE

**Usage**

```fortran
CALL PACKTIMEQQ
  (TIMEDATE, IYR, IMON, IDAY, IHR, IMIN, ISEC)
```

**TIMEDATE**
- INTEGER(4). Packed time and date information.

**IYR**
- INTEGER(2). Year (xxxx AD).

**IMON**
- INTEGER(2). Month (1 - 12).

**IDAY**
- INTEGER(2). Day (1 - 31)
IHR INTEGER(2). Hour (0 - 23)
IMIN INTEGER(2). Minute (0 - 59)
ISEC INTEGER(2). Second (0 - 59)

The packed time is the number of seconds since 00:00:00 Greenwich mean time, January 1, 1970. You can numerically compare packed time items. You can use PACKTIMEQQ to work with relative date and time values. Use UNPACKTIMEQQ to unpack time information. SETFILETIMEQQ uses packed time.

Example
USE IFLPORT
INTEGER(2) year, month, day, hour, minute, second, hund
INTEGER(4) timedate
INTEGER(4) y4,m4,d4,h4,s4,hu4
CALL GETDAT (y4, m4, d4)
year = y4
month = m4
day = d4
CALL GETTIM (h4, m4, s4, hu4)
hour = h4
minute = m4
second = s4
hund = hu4
CALL PACKTIMEQQ (timedate, year, month, day, hour, & minute, second)
END
PEEKCHARQQ

Checks the keystroke buffer for a recent console keystroke.

Prototype

USE IFLPORT
or
INTERFACE
! TEST FOR CONSOLE INPUT
   LOGICAL(4) FUNCTION PEEKCHARQQ()
   END FUNCTION PEEKCHARQQ
END INTERFACE

Description

Checks the keystroke buffer for a recent console keystroke and returns .TRUE. if there is a character in the buffer or .FALSE. if there is not.

Usage

result = PEEKCHARQQ ( )

Results

The result type is LOGICAL(4). The result is .TRUE. if there is a character waiting in the keyboard buffer; otherwise, .FALSE..

To find out the value of the key in the buffer, call GETCHARQQ. If there is no character waiting in the buffer when you call GETCHARQQ, GETCHARQQ waits until there is a character in the buffer. If you call PEEKCHARQQ first, you prevent GETCHARQQ from halting your process while it waits for a keystroke. If there is a keystroke, GETCHARQQ returns it and resets PEEKCHARQQ to .FALSE..

Example

USE IFLPORT
LOGICAL(4) pressed / .FALSE. /
DO WHILE (.NOT. pressed)
    WRITE(*,*) ' Press any key'
    pressed = PEEKCHARQQ ( )
END DO
END

**PERROR**

* Sends a message to standard error

**Prototype**

```plaintext
INTERFACE
    SUBROUTINE PERROR (STRING)
        CHARACTER(LEN=*) STRING
    END SUBROUTINE PERROR
END INTERFACE
```

**Description**

Sends a message to the standard error stream, preceded by the specified `STRING`.
POPCNT

Counts the number of 1-bits in the given value

Prototype

INTERFACE
   INTEGER(4) FUNCTION POPCNT (VALUE)
   INTEGER(4), INTENT(IN) :: VALUE
END FUNCTION POPCNT
END INTERFACE

VALUE             a positive integer value

Description
This function counts the number of 1-bits in the given value.

Output
Number of 1-bits.

POPPAR

Population parity

Prototype

INTERFACE
   INTEGER(4) FUNCTION POPPAR (P)
   TYPE P
   END FUNCTION POPPAR
END INTERFACE
**TYPE**

- may have the following types: BYTE, INTEGER, INTEGER(2), INTEGER(4), LOGICAL(1), LOGICAL(2), LOGICAL(4), REAL(4), POINTER

**P**

- any scalar value up to a maximum of 32 bits in length

**Description**

This function returns 0 if the number of bits in the argument is even, 1 if the number is odd.

---

**PUTC**

*Writes a character to standard output.*

**Prototype**

```fortran
INTERFACE
  INTEGER(4) FUNCTION PUTC(CH)
  CHARACTER(LEN=1) CH
END FUNCTION PUTC
END INTERFACE
```

**CH**

- a character variable

**Description**

Writes a character to the standard output device. Intel Fortran assumes that external unit 6 is connected to the standard output device (stdout), which is where your output will be sent. Typically stdout is your terminal screen. If unit 6 is connected to some other device, this routine will still send output to the standard output device.

**Output**

- A zero if successful; otherwise, an error code.
QRANSET

*Sets the seed for a sequence of pseudo-random numbers.*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE QRANSET (RSEED)
    !MS$ATTRIBUTES ALIAS:'qranset_'::QRANSET
    REAL(16), INTENT(IN) :: RSEED
  END SUBROUTINE
END INTERFACE
```

**Description**

Sets the seed for a sequence of pseudo-random numbers.

**Output**

A changed seed.

QSORT

*Sorts an array*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE QSORT(ARRAY, LEN, ISIZE, COMP)
    TYPE(ARRAY(LEN))
    INTEGER LEN, ISIZE
  INTERFACE
    INTEGER(2) FUNCTION COMP(P1, P2)
      TYPE P1, P2
  END INTERFACE
END INTERFACE
```
END FUNCTION COMP
END INTERFACE
END SUBROUTINE QSORT
END INTERFACE

ARRAY is a one-dimensional array of any of the following
TYPE is any intrinsic or derived type
LEN is the number of elements in the array
ISIZE is the size in bytes of a single element of the array
COMP is a comparison function that you must supply that
returns
<0, if P1 .LT. P2
=0, if P1 = P2
>0, if P1 .GT. P2

Description
Sorts the given array using the given comparison function.

NOTE. QSORT can provide unpredictable results for multi-dimensional arrays. It assumes C language element order and one-dimensional arrays.

Output
QSORT returns your array sorted in place in ascending order.

RAISEQQ
Sends a signal to the executing program.

Prototype
USE IFLPORT

or

INTERFACE
  INTEGER(4) FUNCTION RAISEQQ(SIGNUMBER)
  INTEGER(4) SIGNUMBER
END FUNCTION
END INTERFACE

Usage
result = RAISEQQ (SIGNUMBER)
SIGNUMBER the number of the signal to raise. One of the following constants (defined in IFLPORT.F90)
SIG$ABORT Abnormal termination
SIG$FPE Floating-point error
SIG$ILL Illegal instruction
SIG$INT CTRL+C signal
SIG$SEGV Illegal storage access
SIG$TERM Termination request

If you do not install a signal handler (with SIGNALQQ, for example), when a signal occurs the system by default terminates the program.

Results
The result is zero if successful; otherwise, nonzero.

If a signal-handling routine for SIGNUMBER has been installed by a prior call to SIGNALQQ, RAISEQQ causes that routine to be executed. If no handler routine has been installed, the system terminates the program (the default action).
RAN

Generates a random number between 0 and 1.

Prototype

INTERFACE
    REAL(4) FUNCTION RAN(ISEED)
    !MS$ATTRIBUTES ALIAS:’ran_’:::RAN
    INTEGER(4), INTENT(IN):::ISEED

END FUNCTION RAN
END INTERFACE

ISEED must be an INTEGER(4) variable or array element.
RAN stores a number in ISEED to be used by the next call to RAN. ISEED should initially be set to an odd number, preferably very large; see the Example.

Description

Generates random numbers between 0 and 1 from an integer seed ISEED. It updates ISEED with the new value of the internal seed. At the first call, it is recommended to initialize ISEED to a large, odd value.

Class

Elemental nonstandard function.

Example

INTEGER(4) iseed
REAL(4) rnd
iseed = 425001
rnd = RAN(iseed)

NOTE. To ensure different random values for each run of a program, ISEED should be set to a different value each time the program is run. One way to implement this would be to have the user enter the seed at the start of the program. Another way would be to compute a value from the current year, day, and month (returned by IDATE) and the number of seconds since midnight.

RAND
Generates a random number in the range of 0.0 to 1.0.

Prototype

INTERFACE
   REAL(4) FUNCTION RAND(ISEED)
   INTEGER(4), INTENT(IN):: ISEED !New SEED
END FUNCTION RAND
REAL(4) FUNCTION RAND2()
END FUNCTION RAND2()
END INTERFACE

Description
Generate a random number uniformly distributed in the range of 0.0 to 1.0. At the first call, it is recommended to set ISEED to a large, odd value to initialize the internal seed. ISEED is not updated. For subsequent calls, ISEED should be set to zero to get the next random number in the sequence. If ISEED is set to one, the internal seed can also be set by
CALL SEED(ISEED).
When RAND called without arguments, ISEED assumed to be 0.

Class
Elemental nonstandard function.

Result Type and Parameter Type
REAL(4) type.

Output
The output depends on the parameter value as follows:
• If the input parameter is equal to zero, RAND returns the next number in the pseudorandom sequence.
• If the input parameter is equal to one, RAND restarts the pseudorandom number sequence and returns the first number of the sequence.
• If the input parameter is greater than one, the value is used as the seed for a new pseudorandom sequence, and RAND returns the first number in that sequence.

Example
INTERFACE
    REAL(4) FUNCTION RAND(ISEED)
    INTEGER(4) ISEED
END INTERFACE
INTEGER(4) ISEED
REAL(4) rv
rv = RAND(ISEED)
END

NOTE. For details about restarting the pseudorandom number generator used by IRAND and RAND, see the “SRAND” section.
**RANDOM**

*Random number generator*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE RANDOM (R)
    REAL (4) R
  END SUBROUTINE RANDOM
END INTERFACE
```

**Description**

RANDOM generates a pseudo-random number, based upon the value of the seed set by the RANSET function. Portability functions RANDOM, RAND, and DRAND all generate the same results. A given seed will always generate the same sequence of pseudo-random numbers. RANDOM is an implementation of the algorithm described in *Random Number Generators: Good ones are hard to find*, by Park, S.K., and Miller, K.W., in Comm ACM, Oct. 1988, 1192-1201. This is also described in section 7 of Numerical Recipes. This routine is provided for compatibility with legacy FORTRAN programs. You should use the RANDOM_NUMBER and RANDOM_SEED intrinsic functions that are standard Fortran when possible.

**Output**

Random real number.
RANDU

Generates a pseudorandom number in the range 0.0 to 1.0.

Prototype

INTERFACE
  REAL(4) FUNCTION RANDU (IL, I2, X)
  END FUNCTION RANDU
END INTERFACE

IL, I2 must be INTEGER(2) variables or array elements that contain the SEED for computing the random number. These values are updated during the computation so that they contain the updated seed.

X A REAL(4) variable or array element where the computed random number is returned.

Result

The result is returned in X, which must be of type REAL(4). The result value is a pseudorandom number in the range 0.0 to 1.0.

The algorithm for computing the random number value is based on the values for IL and I2.

If IL=0 and I2=0, the generator base is set as follows:

\[ X(N + 1) = 2^{16} + 3 \]

Otherwise, it is set as follows:

\[ X(N + 1) = (2^{16} + 3) \times X(N) \mod 2^{32} \]

The generator base \(X(N + 1)\) is stored in IL, I2.

The result is \(X(N + 1)\) scaled to a real value \(Y(N + 1)\), for \(0.0 \leq Y(N + 1) < 1.0\).
Example

REAL X
INTEGER(2) I, J
...
CALL RANDU (I, J, X)

If I and J are values 4 and 6, X stores the value 5.4932479E-04.

**RANF**

*Generates a random number between 0. and RAND_MAX.*

**Prototype**

INTERFACE
  REAL(4) FUNCTION RANF ()
  END FUNCTION RANF
END INTERFACE

**Description**

RANF returns a single-precision pseudo-random number between 0.0 and RAND_MAX as defined in the C library, normally 0x7FFF 2^{15}. The initial seed is set by

CALL SRAND(ISEED)

**Output**

A random number of the type REAL(4). ISEED is of INTEGER(4).
RANGET

*Returns the current seed*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE RANGET (S)
  INTEGER(4) S
  END SUBROUTINE RANGET
END INTERFACE
```

**Description**

RANGET returns the current seed used for a sequence of pseudo-random numbers.

**Output**

The internal seed. This routine is not thread-safe.

RANSET

*Sets the seed for the random number generator*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE RANSET (ISEED)
  REAL(4) ISEED
  END SUBROUTINE RANSET
END INTERFACE
```
Description
Sets the seed for a sequence of pseudo-random numbers.

Output
A changed seed.

RENAME
Renames a file

Prototype
INTERFACE
   FUNCTION RENAME (FROM, TO)
     CHARACTER (LEN=*) FROM, TO
   END FUNCTION RENAME
END INTERFACE

FROM       a character path name of the origin file
TO         a character path name that specifies the name and
           location where you wish to place the FROM file

Description
This routine renames a file. Either FROM or TO may include a fully qualified
or relative path name. RENAME accepts either forward or backward slashes
as directory separators, and converts them to the form appropriate for the
host operating system. On Windows, you may include drive letters in the path
also. Drive letters are not accepted on Linux systems.
The FROM file must exist when rename is called, but you do not have to have it connected to a logical unit.

**NOTE.** The FROM and TO paths for the files must be on the same physical device. You cannot use this routine to move a file from one device to another.

This routine is thread-safe, and locks the associated stream before I/O is performed.

**Output**
A zero status is returned for success, non-zero for failure.

### RENAMEFILEQQ
* Renames a file.

#### Prototype

```fortran
USE IFLPORT
or
INTERFACE
  LOGICAL(4) FUNCTION RENAMEFILEQQ(OLDNAME,NEWNAME)
  CHARACTER(LEN=*) OLDNAME,NEWNAME
  END FUNCTION RENAMEFILEQQ
END INTERFACE
```

#### Usage

```fortran
result = RENAMEFILEQQ (OLDNAME,NEWNAME)
OLDNAME CHARACTER(LEN=*). File to be renamed.
NEWNAME CHARACTER(LEN=*). New name of the file to be renamed.
```
Results

The result is .TRUE. if successful; otherwise, .FALSE..

You can use RENAMEFILEQQ to move a file from one directory to another on the same drive by giving a different path in the NEWNAME parameter.

If the function fails, you should call GETLASTERRORQQ to determine the reason. One of the following errors can be returned:

- ERR$ACCES: Permission denied. The file's permission setting does not allow the specified access.
- ERR$EXIST: The file already exists.
- ERR$NOENT: File or path specified by OLDNAME not found.
- ERR$XDEV: Attempt to move a file to a different device.

Example

USE iflport
INTEGER(4) len
CHARACTER(80) oldname, newname
LOGICAL(4) result
WRITE(*,'(A)') ' Enter old name: '
len = GETSTRQQ(oldname)
WRITE(*,'(A)') ' Enter new name: '
len = GETSTRQQ(newname)
result = RENAMEFILEQQ(oldname, newname)
END

RINDEX

Locates the index of the last occurrence
of a substring within a string

Prototype

INTERFACE
INTEGER(4) FUNCTION RINDEX (S1, S2)
    CHARACTER(LEN=*) S1, S2
END FUNCTION RINDEX
END INTERFACE

S1           original string to search
S2           string to search for

Description
This function locates the index of the last occurrence of a substring within a string.

Output
Starting position of the final occurrence of S2 in S1.

RTC
Returns the number of seconds elapsed since a specific Greenwich mean time.

Prototype
USE IFLPORT

Usage
result = RTC()

Results
The result is of type REAL(8). The result is the number of seconds elapsed since 00:00:00 GMT, January 1, 1970.

Example
USE IFLPORT
REAL(8) S, S1, TIME_SPENT
INTEGER(4) I, J
S = RTC ()
CALL SLEEP (4)
S1 = RTX()
TIME_SPENT = S1 - S
PRINT *, 'It took ', TIME_SPENT, 'seconds to run.'

**RUNQQ**

*Executes another program and waits for it to complete.*

**Prototype**

```fortran
USE IFLPORT
or
INTEGER(2) FUNCTION RUNQQ(PROGNAME,COMMANDLINE)
   CHARACTER(LEN=*) PROGNAME, COMMANDLINE
END FUNCTION
```

**Usage**

result = RUNQQ (FILENAME, COMMANDLINE)

FILENAME    Input. CHARACTER (LEN=*) . Filename of a program to be executed.
COMMANDLINE Input. CHARACTER (LEN=*) . Command-line arguments passed to the program to be executed.

**Results**

The result type is INTEGER(2). If the program executed with RUNQQ terminates normally, the exit code of that program is returned to the program that launched it. If the program fails, -1 is returned.
The **RUNQQ** function executes a new process for the operating system using the same path, environment, and resources as the process that launched it. The launching process is suspended until execution of the launched process is complete.

**Example**

```plaintext
USE IFLPORT
INTEGER(2) result
result = RUNQQ('dir', '/Os')
END
```

---

**SCWRQQ**

*Returns the floating-point processor control word.*

**Prototype**

```plaintext
USE IFLPORT
or

INTERFACE
SUBROUTINE SCWRQQ(CONTROL)
    INTEGER(2) CONTROL
END SUBROUTINE
END INTERFACE
```

**Usage**

```plaintext
CALL SCWRQQ (CONTROL)
CONTROL Output. INTEGER (2). Floating-point processor control word.

SCRWQQ performs the same function as GETCONTROLFPQQ, and is provided for compatibility.
```
**SCANENV**

*Scans the environment for environment variable.*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE SCANENV (ENVNAME, ENVTEXT, ENVVALUE)
    CHARACTER (LEN=*) INTENT(IN) :: ENVNAME
    CHARACTER (LEN=*) INTENT(OUT) :: ENVTEXT, ENVVALUE
  END SUBROUTINE SCANENV
END INTERFACE
```

**ENVNAME**

a CHARACTER variable containing the name of an environment variable you need to find the value for.

**ENVTEXT**

set to the full text of the environment variable found, or to a ' ' if nothing is found.

**ENVVALUE**

set to the value associated with the environment found or ' ' if nothing is found.

**Description**

Scans the environment for an environment variable that matches ENVNAME and returns the value or string it is set to.

**Output**

The text string or the value of the environment variable.
SEED

Sets the starting point for the random number generator

Prototype

INTERFACE
  SUBROUTINE SEED (ISEED)
  INTEGER(4) ISEED
  END SUBROUTINE SEED
END INTERFACE

Description

Sets the internal seed for a sequence of pseudo-random numbers.

Output

A changed seed. This routine is not thread-safe.

SECNDS

Returns the number of seconds since last call

Prototype

INTERFACE
  REAL(4) FUNCTION SECNDS(TIME)
  REAL(4) TIME
  END FUNCTION SECNDS
END INTERFACE

TIME 0.0 to start the clock, or the last value returned from SECNDS
Description
This function returns the elapsed time in seconds since the last call to SECNDS, or the number of seconds since midnight if the parameter is 0.0.

Example

```fortran
PROGRAM TIMEIT
REAL STARTTIME,STOPTIME
STARTTIME= SECNDS(0.0)
DO 10 I = 1,100000
   I = I +1
10 CONTINUE
STOPTIME= SECNDS(STARTTIME)
PRINT *, 'Elapsed time was: ',STOPTIME
END
```

We provide this routine for compatibility with other FORTRAN compilers. You can time a section of your code's execution, or time your whole program. You can also time sections of your program and add the times together. This routine is primarily useful in benchmarking, or as a rough profiling guide of where your application spends most of its execution time.

To start the timing clock, call SECNDS with 0.0, and save the result in a local variable. To get the elapsed time since the last call to SECNDS, pass the local variable to SECNDS on the next call.

This routine is thread-safe.

Output
The elapsed time in whole seconds since midnight, or since midnight minus the value of the supplied floating point argument. If you want more accurate timing, see the subroutine DCLOCK.
SETCONTROLFPQQ

Sets the value of the floating-point processor control word.

Prototype

USE IFLPORT
or
INTERFACE
  SUBROUTINE SETCONTROLFPQQ(CONTROL)
      INTEGER(2) CONTROL
  END SUBROUTINE
END INTERFACE

Usage

CALL SETCONTROLFPQQ (CONTROLWORD)
CONTROLWORD INTEGER(2). Floating-point processor control word.

The floating-point control word allows you to specify how various exception conditions are handled by the floating-point math co-processor. You can also set the floating-point precision, and specify the floating-point rounding mechanism used.

The IFLPORT.F90 module file contains constants defined for the control word as follows:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPCW$MCW_IC</td>
<td>Z’1000’</td>
<td>Infinity control mask</td>
</tr>
<tr>
<td>FPCW$AFFINE</td>
<td>Z’1000’</td>
<td>Affine infinity</td>
</tr>
<tr>
<td>FPCW$PROJECTIVE</td>
<td>Z’0000’</td>
<td>Projective infinity</td>
</tr>
<tr>
<td>FPCW$MCW_PC</td>
<td>Z’0300’</td>
<td>Precision control mask</td>
</tr>
<tr>
<td>FPCW$64</td>
<td>Z’0300’</td>
<td>64-bit precision</td>
</tr>
<tr>
<td>FPCW$53</td>
<td>Z’0200’</td>
<td>53-bit precision</td>
</tr>
</tbody>
</table>
Thedefaultsforthefloating-pointcontrolwordare53-bitprecision,round
tonearest,andthedenormal,underflowandinexactprecisionexceptions
disabled. An exception is disabled if its flag is set to 1 and enabled if its flag
iscleared to 0.

Setting the floating-point precision and rounding mechanism can be useful
ifyouarereusingoldcodethatis sensitive to the floating-point precision
standard used and you want to get the same results as on the old machine.

You can use GETCONTROLFPQQ to retrieve the current control word and
SETCONTROLFPQQ to change the control word. If you need to change the
control word, always use SETCONTROLFPQQ to make sure that special
routines handling floating-point stack exceptions and abnormal propagation
work correctly.

Example
USE IFLPORT
INTEGER(2) status, control, controlo

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Hex Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPCW$24</td>
<td>Z’0000’</td>
<td>24-bit precision</td>
</tr>
<tr>
<td>FPCW$MCW_RC</td>
<td>Z’0C00’</td>
<td>Rounding control mask</td>
</tr>
<tr>
<td>FPCW$CHOP</td>
<td>Z’0C00’</td>
<td>Truncate</td>
</tr>
<tr>
<td>FPCW$UP</td>
<td>Z’0800’</td>
<td>Round up</td>
</tr>
<tr>
<td>FPCW$DOWN</td>
<td>Z’0400’</td>
<td>Round down</td>
</tr>
<tr>
<td>FPCW$NEAR</td>
<td>Z’0000’</td>
<td>Round to nearest</td>
</tr>
<tr>
<td>FPCW$MCW_EM</td>
<td>Z’003F’</td>
<td>Exception mask</td>
</tr>
<tr>
<td>FPCW$INVALID</td>
<td>Z’0001’</td>
<td>Allow invalid numbers</td>
</tr>
<tr>
<td>FPCW$DENORMAL</td>
<td>Z’0002’</td>
<td>Allow denormals (very small numbers)</td>
</tr>
<tr>
<td>FPCW$ZERODIVIDE</td>
<td>Z’0004’</td>
<td>Allow divide by zero</td>
</tr>
<tr>
<td>FPCW$OVERFLOW</td>
<td>Z’0008’</td>
<td>Allow overflow</td>
</tr>
<tr>
<td>FPCW$UNDERFLOW</td>
<td>Z’0010’</td>
<td>Allow underflow</td>
</tr>
<tr>
<td>FPCW$INEXACT</td>
<td>Z’0020’</td>
<td>Allow inexact precision</td>
</tr>
</tbody>
</table>
CALL GETCONTROLFPQQ(control)
WRITE (*,9000) 'Control word: ', control
! Save old control word controlo = control
! Clear all flags control = control .AND. Z’0000’
! Set new control to round up
control = control .OR. FPCW$UP
CALL SETCONTROLFPQQ(control)
CALL GETCONTROLFPQQ(control)
WRITE (*,9000) 'Control word: ', control
9000 FORMAT (1X,A,Z4)
END

SETDAT
Sets the current system date in years, months, and day

Prototype
INTERFACE SETDAT
LOGICAL(4) FUNCTION SETDAT(YEAR,MONTH,DAY)
INTEGER(4) :: YEAR,MONTH,DAY! YEAR IS 4 DIGITS
END FUNCTION
LOGICAL(4) FUNCTION SETDAT_DVF(YEAR,MONTH,DAY)
INTEGER(2) :: YEAR,MONTH,DAY! YEAR is 4 digits
END FUNCTION
END INTERFACE

YEAR 4-digit integer
MONTH between 1-12
DAY between 1-31
**Description**
This subroutine sets the current system date in years, months, and day. If the values are not valid, the date is not set.

**Output**
Changed date on the system executing the program.

---

**SETENVQQ**

Sets the value of an existing environment variable.

**Prototype**

```fortran
USE IFLPORT
or
LOGICAL(4) FUNCTION SETENVQQ(INPUT_STRING)
    CHARACTER(LEN=*) INPUT_STRING
END FUNCTION SETENVQQ
```

**Description**

Sets the value of an existing environment variable, or adds and sets a new environment variable.

**Usage**

```fortran
result = SETENVQQ (INPUT_STRING)
```

`INPUT_STRING` CHARACTER(LEN=*). String containing both the name and the value of the variable to be added or modified. Must be in the form: `VARNAME = VALUE`, where `VARNAME` is the name of an environment variable and `value` is the value being assigned to it.
Results

The result is .TRUE. if successful; otherwise, .FALSE..

Environment variables define characteristics of the environment in which a program executes. For example, the LIB environment variable defines the default search path for libraries to be linked with a program.

`SETENVQQ` deletes any terminating blanks in `INPUT_STRING`. Although the equal sign (=) is an illegal character within an environment value, you can use it to terminate `VALUE` so that trailing blanks are preserved. For example, the string `PATH= =` sets `VALUE` to ".

You can use `SETENVQQ` to remove an existing variable by giving a variable name followed by an equal sign with no value. For example, `LIB=` removes the variable `LIB` from the list of environment variables. If you specify a value for a variable that already exists, its value is changed. If the variable does not exist, it is created.

`SETENVQQ` affects only the environment that is local to the current process. You cannot use it to modify the command-level environment. When the current process terminates, the environment reverts to the level of the parent process. In most cases, this is the operating system level. However, you can pass the environment modified by `SETENVQQ` to any child process created by `RUNQQ`, `PXFFORK`, or other means of creating a child process. These child processes get new variables and/or values added by `SETENVQQ`.

Example

```
USE IFLPORT
! Note, compile this example with /nbs
LOGICAL(4) success
success = SETENVQQ("PATH=c:\users\mjsmith\bin")
success = &
  SETENVQQ("LIB=c:\program
  files\intel\compiler4.5\lib")
PRINT *, SUCCESS
END
```
SETERRORMODEQQ

Sets the prompt mode for critical errors.

Prototype

USE IFLPORT
or

INTERFACE
  SUBROUTINE SETERRORMODEQQ(PROMPT)
    LOGICAL(4) PROMPT
  END SUBROUTINE
END INTERFACE

Description

Sets the prompt mode for critical errors that by default generate system prompts.

Usage

CALL SETERRORMODEQQ (PROMPT)
PROMPT LOGICAL(4). PROMPT determines whether a prompt is displayed when a critical error occurs.

Certain I/O errors cause the system to display an error prompt. For example, attempting to write to a disk drive with the drive door open generates an "Abort, Retry, Ignore" message. When your program begins execution, system error prompting is enabled by default. You can enable system error prompts by calling SETERRORMODEQQ with PROMPT set to ERR$HARDPROMPT (defined in IFLPORT.F90).

If you disable prompting, serious I/O errors that would normally result in a prompt are silent.
Errors in I/O statements such as OPEN, READ, and WRITE fail immediately instead of being interrupted with prompts. This gives you more direct control of what happens when there is an error. You can use the ERR= facility of many I/O statements to give a label to branch to for error handling.

You can turn off prompt mode by setting PROMPT to .FALSE. or to the constant ERR$HARDFAIL (defined in IFLPORT.F90). You should be aware that SETERRORMODEQQ affects only errors that generate a system prompt. It does not affect other I/O errors, such as writing to a nonexistent file or attempting to open a nonexistent file with STATUS='OLD'.

Example 1

! PROGRAM 1
!  DRIVE B door open
!  Note: you should compile these examples with /nbs
OPEN (10, FILE = 'B:\NOFILE.DAT', ERR = 100)
!  Generates a system prompt error here and waits for
!  the user to respond to the prompt before continuing
100 WRITE(*,*) ' Continuing'
END

Example 2

! PROGRAM 2
!  DRIVE B door open
!  Note: you should compile these examples with /nbs
USE IFLPORT
CALL SETERRORMODEQQ(.FALSE.)
OPEN (10, FILE = 'B:\NOFILE.DAT', ERR = 100)
!  Causes the statement at label 100 to execute
!  without system prompt
100 WRITE(*,*) ' Drive B: not available, opening
& &alternative drive.'
 OPEN (10, FILE = 'C:\NOFILE.DAT')
END
SETFILEACCESSQQ

Sets the file access mode for a specified file.

Prototype

USE IFLPORT

Usage

result = SETFILEACCESSQQ (FILENAME, ACCESS)

FILENAME    Input. CHARACTER (LEN=*). Name of the file to set access for.
ACCESS      Input. INTEGER (4). Constant that sets the access. Can be any combination of the following flags, combined by an inclusive OR (such as IOR or OR):

FILE$ARCHIVE  Marked as having been copied to a backup device.
FILE$HIDDEN    Hidden. The file does not appear in the directory list that you can request from the command console.
FILE$NORMAL    No special attributes (default).
FILE$READONLY  Write-protected. You can read the file, but you cannot make changes to it.
FILE$SYSTEM    Used by the operating system

The flags are defined in module INCLUDE\DFLIB.F90.

Results

The result is of type LOGICAL (4). The result is .TRUE. if successful; otherwise, .FALSE..

To set the access value for a file, add the constants representing the appropriate access.
Example
USE IFLPORT
INTEGER(4) PERMIT
LOGICAL(4) RESULT
PERMIT = 0 ! clear permit
PERMIT = IOR(FILE$READONLY, FILE$HIDDEN)
RESULT = SETFILEACCESSQQ ('formula.f90', PERMIT)
END

SETFILETIMEQQ
Sets the modification time for a specified file.

Prototype
USE IFLPORT
or:
IA-32 systems:
INTERFACE
  LOGICAL(4) FUNCTION SETFILETIMEQQ(NAME, TIMEDATE)
  CHARACTER(LEN=*) NAME
  INTEGER(4) TIMEDATE
END FUNCTION
END INTERFACE

Itanium®-based systems:
INTERFACE
  LOGICAL(4) FUNCTION SETFILETIMEQQ(NAME, TIMEDATE)
  CHARACTER(LEN=*) NAME
  INTEGER(8) TIMEDATE
END FUNCTION
END INTERFACE
Usage

result = SETFILETIMEQQ (NAME, TIMEDATE)

NAME CHARACTER(LEN=*) . Name of a file.
TIMEDATE INTEGER(4) . Time and date information, as packed by PACKTIMEQQ.

Results

The result is .TRUE. if successful; otherwise, .FALSE..

The modification time is the time the file was last modified. The process that calls SETFILETIMEQQ must have write access to the file; otherwise, you cannot change the time. If you set TIMEDATE to FILE$CURTIME (defined in IFLPORT.F90), SETFILETIMEQQ sets the modification time to the current system time.

If the function fails, call GETLASTERRORQQ to determine the reason, which can be one of the following:

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR$ACCES</td>
<td>Permission denied. The file's (or directory's) permission setting does not allow the specified access.</td>
</tr>
<tr>
<td>ERR$INVAL</td>
<td>Invalid argument; TIMEDATE argument is invalid.</td>
</tr>
<tr>
<td>ERR$MFILE</td>
<td>Too many open files (the file must be opened to change its modification time).</td>
</tr>
<tr>
<td>ERR$NOENT</td>
<td>File or path not found.</td>
</tr>
<tr>
<td>ERR$NOMEM</td>
<td>Not enough memory is available to execute the command; or the available memory has been corrupted; or an invalid block exists, indicating that the process making the call was not allocated properly.</td>
</tr>
</tbody>
</table>

Example

USE IFLPORT
INTEGER(2) day, month, year
INTEGER(2) hour, minute, second, hund
INTEGER(4) timedate
INTEGER(4) hr1, mn1, se1, hu1, ye1, mo1, da1
LOGICAL(4) result
CALL GETDAT(ye1, mo1, da1)
  year = ye1
  month = mo1
  day = da1
CALL GETTIM(hr1, mn1, se1, hu1)
  hour = hr1
  minute = mn1
  second = se1
  hund = hu1
CALL PACKTIMEQQ(timedate, year, month, day, &
  hour, minute, second)
result = SETFILETIMEQQ('myfile.dat', timedate)
PRINT *, RESULT
END

SETTIM

Sets the current system date in years, months, and days

Prototype

INTERFACE SETTIM
  LOGICAL(4) FUNCTION SETTIM (HOUR, MINUTE, &
  SECOND, HUNDRETH)
  INTEGER(4), INTENT(IN) :: HOUR, MINUTE, &
  SECOND, HUNDRETH
END FUNCTION

LOGICAL(4) FUNCTION SETTIM_DVF (HOUR, MINUTE, &
  SECOND, HUNDRETH)
  INTEGER(2), INTENT(IN) :: HOUR, MINUTE, &
  SECOND, HUNDRETH

END

Portability Functions
HOUR between 0 - 23
MINUTE between 0 - 59
SECOND between 0 - 59
HUNDREDTH between 0 - 99

Description
This subroutine sets the current system time in hours, minutes, seconds and hundredths of a second. If the values are not valid, the time is not set.

Output
Changed current time on the system executing the program.

**SHIFTL**

*Shifts a value to the left by a specified number of bit positions*

**Prototype**

```fortran
INTERFACE
   INTEGER(4) FUNCTION SHIFTL(IVALUE,ISHIFTCOUNT)
   INTEGER(4) IVALUE,ISHIFTCOUNT
END INTEGER FUNCTION SHIFTL
END INTERFACE
```

- **IVALUE** an integer value
- **ISHIFTCOUNT** the number of bit positions to shift. Must be positive
**Description**

This is an arithmetic shift. A shift count greater than the size in bits of the input value returns a result of zero.

This routine is thread-safe.

**Output**

The input value is shifted left by ISHIFTCOUNT bit positions.

---

**SHIFTR**

*Shifts a value to the right by a specified number of bit positions*

**Prototype**

```fortran
INTERFACE
   INTEGER(4) FUNCTION SHIFTR(IVALUE,ISHIFTCOUNT)
   INTEGER(4) IVALUE,ISHIFTCOUNT
END INTEGER FUNCTION SHIFTR
END INTERFACE
```

**IVALUE** an integer value

**ISHIFTCOUNT** the number of bit positions to shift. Must be positive

**Description**

This is an arithmetic shift. A shift count greater than the size in bits of the input value returns a result of zero.

This routine is thread-safe.

**Output**

The input value is shifted right by ISHIFTCOUNT bit positions.
SHORT

Converts an INTEGER(4) value into an equivalent INTEGER(2) type.

USE IFLPORT

Syntax
result = SHORT (INT4)
INT4 Input. INTEGER(4). Value to be converted.

Output
The result is of type INTEGER(2). The result is equal to the lower 16 bits of INT4. If the INT4 value is greater than 32,767, the converted INTEGER(2) value is not equal to the original.

Example
USE IFLPORT
INTEGER(4) INT4
INTEGER(2) INT2
READ (*,*) INT4
INT2 = SHORT (INT4)
WRITE (*, 10) INT4, INT2
SIGNAL

**Controls interrupt signal handling:** changes the action for a specified signal.

USE IFLPORT

**Syntax**

```plaintext
result = SIGNAL (SIGNUM, PROC, FLAG)
```

**SIGNUM**
Input, INTEGER(4): Number of the signal to change. The numbers and symbolic names are listed in the following table.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABRT</td>
<td>6</td>
<td>Abnormal termination</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>8</td>
<td>Floating-point error</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>9</td>
<td>Kill process</td>
</tr>
<tr>
<td>SIGILL</td>
<td>4</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>SIGINT</td>
<td>2</td>
<td>CTRL + C signal</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>11</td>
<td>Illegal storage access</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>15</td>
<td>Termination request</td>
</tr>
</tbody>
</table>

**PROC**
Input. Name of a signal processing routine. It must be declared EXTERNAL. This routine is called only if FLAG is negative.

**FLAG**
Input, INTEGER(4): If negative, the user’s PROC routine is called. If 0, the signal retains its default action; if 1, the signal should be ignored.
Output

The result is of type INTEGER(4). The result is the previous value of PROC associated with the specified signal. For example, if the previous value of PROC was SIG_IGN, the return value is also SIG_IGN. You can use this return value in subsequent calls to SIGNAL if the signal number supplied is invalid, if the flag value is greater than 1, or to restore a previous action definition.

A return value of SIG_ERR indicates an error, in which case a call to IERRNO returns EINVAL. If the signal number supplied is invalid, or if the flag value is greater than 1, SIGNAL returns -(EINVAL) and a call to IERRNO returns EINVAL.

An initial signal handler is in place at startup for SIGFPE (signal 8); its address is returned the first time SIGNAL is called for SIGFPE. No other signals have initial signal handlers.

Be careful when you use SIGNALQQ or the C signal function to set a handler, and then use the Portability SIGNAL function to retrieve its value. If SIGNAL returns an address that was not previously set by a call to SIGNAL, you cannot use that address with either SIGNALQQ or C’s signal function, nor can you call it directly. You can, however, use the return value from SIGNAL in a subsequent call to SIGNAL. This allows you to restore a signal handler, no matter how the original signal handler was set.

All signal handlers are called with a single integer argument, that of the signal number actually received. Usually, when a process receives a signal, it terminates. With the SIGNAL function, a user procedure is called instead. The signal-handler routine must accept the signal number integer argument, even if it does not use it. If the routine does not accept the signal number argument, the stack will not be properly restored after the signal handler has executed.

Because signal-handler routines are usually called asynchronously when an interrupt occurs, it is possible that your signal-handler function will get control when a run-time operation is incomplete and in an unknown state. There are certain restrictions as to which functions you can use in your signal-handler routine:

- Do not do either low-level (such as FGETC) or high-level (such as READ) I/O.
• Do not call heap routines or any routine that uses the heap routines (such as `MALLOC` and `ALLOCATE`).

• Do not use any function that generates a system call (such as `TIME`).

`SIGKILL` can be neither caught nor ignored.

The default action for all signals is to terminate the program with exit code.

The `ABORT` function does not assert the `SIGABRT` signal. The only way to assert `SIGABRT` or `SIGTERM` is to use the `KILL` function.

The `SIGNAL` function can be used to catch the `SIGFPE` exceptions, but it cannot be used to access the error code that caused the `SIGFPE`. To do this, use `SIGNALQQ` instead.

**Example**

```fortran
USE IFLPORT
EXTERNAL H_ABORT
INTEGER(4) IRET1, IRET2, PROCNUM
IRET1 = SIGNAL (SIGABRT, H_ABORT, -1)
WRITE (*,*) 'Set signal handler. Return = ', IRET1
IRET2 = KILL (PROCNUM, SIGABRT)
WRITE (*,*) 'Raised signal. Return = ', IRET2
END
!
! Signal handler routine
!
INTEGER(4) H_ABORT (SIGNUM)
INTEGER(4) SIGNUM
WRITE (*,*) 'In signal handler for SIG$ABORT'
WRITE (*,*) 'SIGNUM = ', SIGNUM
H_ABORT = 1
END
```
**SIGNALQQ**

Registers the function to be called if an interrupt signal occurs.

**Prototype**

USE IFLPORT
or

**IA-32 systems**

INTERFACE

    INTEGER(4) FUNCTION SIGNALQQ(SIGNAL, HANDLER)
    INTEGER(4) SIGNAL
    INTEGER(4) HANDLER
    EXTERNAL HANDLER

    END FUNCTION

END INTERFACE

**Itanium®-based systems**

INTERFACE

    INTEGER(8) FUNCTION SIGNALQQ(SIGNAL, HANDLER)
    INTEGER(4) SIGNAL
    INTEGER(8) HANDLER
    EXTERNAL HANDLER

    END FUNCTION

END INTERFACE

**Usage**

result = SIGNALQQ (SIGNAL, HANDLER)

SIGNAL INTEGER(4). Interrupt type. You should specify one of the following constants, defined in IFLPORT.F90:

- SIG$ABORT Abnormal termination
- SIG$FPE Floating-point error
Portability Functions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG$ILL</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>SIG$INT</td>
<td>CTRL+C SIGNAL</td>
</tr>
<tr>
<td>SIG$SEGV</td>
<td>Illegal storage access</td>
</tr>
<tr>
<td>SIG$TERM</td>
<td>Termination request</td>
</tr>
</tbody>
</table>

HANDLER CHARACTER (LEN=*) You should give a name of function to be executed on interrupt. The function must exist.

Results

The result is a positive integer if successful; otherwise, -1 (SIG$ERR).

SIGNALQQ establishes the function HANDLER as the handler for a signal of the type specified by SIGNAL. If you do not establish a handler, the program terminates when an interrupt signal occurs.

The argument HANDLER is the name of a function and must be declared with either the EXTERNAL or IMPLICIT statements, or have an explicit interface. A function described in an INTERFACE block is EXTERNAL by default, and does not need to be declared EXTERNAL.

When an interrupt occurs, except a SIG$FPE interrupt, the SIGNAL argument SIG$INT is passed to HANDLER, and then HANDLER is executed.

When a SIG$FPE occurs, the function HANDLER is passed two arguments: SIG$FPE and the floating-point error code (for example, FPE$ZERODIVIDE or FPE$OVERFLOW) which identifies the type of floating-point exception that occurred. The floating-point error codes begin with the prefix FPE$ and are defined in IFLPORT.F90.

If HANDLER returns, the calling process resumes execution immediately after the point where it received the interrupt signal. This is true regardless of the type of signal or operating mode.

Because signal-handler routines are normally called asynchronously when an interrupt occurs, it is possible that your signal-handler function will get control when a run-time operation is incomplete and in an unknown state. Therefore, in a signal handler routine, you should not call heap routines or any routine that uses the heap routines (for example, I/O routines, ALLOCATE, and DEALLOCATE).
To test your signal handler routine you can generate interrupt signals by calling \texttt{RAISEQQ}, which causes your program either to branch to the signal handlers set with \texttt{SIGNALQQ}, or to perform the system default behavior if \texttt{SIGNALQQ} has set no signal handler.

The example below demonstrates a signal handler for \texttt{SIG\$ABORT}.

\textbf{Example}

\begin{verbatim}
! This program shows a signal handler for
! SIG\$ABORT
USE IFLPORT
INTERFACE
  FUNCTION h_abort (signum)
    INTEGER(4) h_abort
    INTEGER(2) signum
  END FUNCTION
END INTERFACE
INTEGER(2) i2ret
INTEGER(4) i4ret
i4ret = SIGNALQQ(SIG\$ABORT, h_abort)
WRITE(*,*) \textquote{Set signal handler. Return ='}, i4ret
i2ret = RAISEQQ(SIG\$ABORT)
WRITE(*,*) \textquote{Raised signal. Return ='}, i2ret
END
!
! Signal handler routine
INTEGER(4) FUNCTION h_abort (signum)
  INTEGER(2) signum
  WRITE(*,*) \textquote{In signal handler for SIG\$ABORT'}
  WRITE(*,*) \textquote{signum ='}, signum
  h_abort = 1
END
\end{verbatim}
SLEEP

Suspends execution of a process for a specified interval

Prototype

INTERFACE
    SUBROUTINE SLEEP (TIME)
    INTEGER(4) TIME
    END SUBROUTINE SLEEP
END INTERFACE

TIME Length of time, in seconds, to suspend the calling process.

Description

This function suspends the execution of a process for a specified interval.

Output

None.

SLEEPQQ

Delays execution of the program for a specified duration.

Prototype

USE IFLPORT
or
INTERFACE
    SUBROUTINE SLEEPQQ (DURATION)
SORTQQ

Sorts a one-dimensional array. The array elements cannot be structures.

Prototype
USE IFLPORT

Usage
CALL SORTQQ (ADRARRAY, COUNT, SIZE)
ADRARRAY Input. INTEGER(4). Address of the array (returned by LOC(X)).
COUNT  Input; output. INTEGER (4). On input, number of elements in the array to be sorted. On output, number of elements actually sorted.

SIZE  Input. INTEGER (4). Positive constant less than 32,767 that specifies the kind of array to be sorted. The following constants, defined in \INCLUDE\DFLIB.F90 specify type and kind for numeric arrays:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type of Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRT$INTEGER1</td>
<td>INTEGER(1)</td>
</tr>
<tr>
<td>SRT$INTEGER2</td>
<td>INTEGER(2) or equivalent</td>
</tr>
<tr>
<td>SRT$INTEGER4</td>
<td>INTEGER(4) or equivalent</td>
</tr>
<tr>
<td>SRT$REAL4</td>
<td>REAL(4) or equivalent</td>
</tr>
<tr>
<td>SRT$REAL8</td>
<td>REAL(8) or equivalent</td>
</tr>
</tbody>
</table>

If the value provided in SIZE is not a symbolic constant and is less than 32,767, the array is assumed to be a character array with SIZE characters per element.

To be certain that SORTQQ is successful, compare the value returned in COUNT to the value you provided. If they are the same, then SORTQQ sorted the correct number of elements.

CAUTION. The location of the array must be passed by address using the LOC (X) function. This defeats Fortran type-checking, so you must make certain that the COUNT and SIZE arguments are correct.

If you pass invalid arguments, SORTQQ attempts to sort random parts of memory. If the memory it attempts to sort is allocated to the current process, that memory is sorted; otherwise, the operating system intervenes, the program is halted, and you get a General Protection Violation message.
Example

! Sort a 1-D array
!
USE IFLPORT
INTEGER(2) ARRAY(10)
INTEGER(2) I
DATA ARRAY /143,99,612,61,712,9112,6,555,2223,67/
! Sort array
CALL SORTQQ (LOC(ARRAY), 10, SRT$INTEGER2)
! Display the sorted array
DO I = 1, 10
   WRITE (*, 9000) I, ARRAY(I)
END DO

SPLITPATHQQ

Breaks a filepath or directory path into its components.

Prototype

USE IFLPORT
or
INTERFACE
   INTEGER(4) FUNCTION SPLITPATHQQ(PATH, DRIVE, DIR, NAME, EXT)
   CHARACTER(LEN=*) NAME, EXT, PATH, DRIVE, DIR
END FUNCTION
END INTERFACE

Usage

result = SPLITPATHQQ (PATH, DRIVE, DIR, NAME, EXT)
PATH CHARACTER (LEN=*) . Path that you want to break into components. Forward slashes (/), backslashes (\), or both can be present in PATH.

DRIVE CHARACTER (LEN=*) . Drive letter designation followed by a colon.

DIR CHARACTER (LEN=*) . Path of directories, including the trailing slash.

NAME CHARACTER (LEN=*) . Name of file or, if no file is specified in PATH, name of the lowest directory. If a filename, does not include an extension.

EXT CHARACTER (LEN=*) . Filename extension, if any, including the leading period (\).

Results
The result is the length of DIR.

PATH can be a complete or partial file specification.

MAXPATH is a symbolic constant defined in module IFLPORT.F90 as 260.

Example
USE IFLPORT
CHARACTER (MAXPATH) buf
CHARACTER (3) drive
CHARACTER (256) dir
CHARACTER (256) name
CHARACTER (256) ext
CHARACTER (256) file
INTEGER (4) length
! Note, this example should be compiled with /nbs
buf = 'b:\fortran\test\runtime\tsplit.for'
length = SPLITPATHQQ(buf, drive, dir, name, ext)
WRITE(*,*) drive, dir, name, ext
file = 'partial.f90'
length = SPLITPATHQQ(file, drive, dir, name, ext)
WRITE(*,*) drive, dir, name, ext
END
SRAND

*Restart the pseudorandom number generator used by IRAND and RAND.*

**Prototype**

```fortran
INTERFACE
   SUBROUTINE SRAND (NEW_SEED)
      INTEGER(4), INTENT(IN) :: NEW_SEED
   END SUBROUTINE SRAND
END INTERFACE
```

**NEW_SEED**  Must be of INTEGER(4) type. The same value for NEW_SEED generates the same sequence of random numbers. To vary the sequence, call SRAND with a different NEW_SEED value each time the program is executed. The default for NEW_SEED is 1.

**Description**

Restart the pseudorandom number generator used by IRAND and RAND. Calling SRAND is equivalent to calling IRAND or RAND with a new seed.

**Class**

Specific nonstandard subroutine.

**Example**

```fortran
! Random number out of 100 between .5 and .6
USE IFLPORT
ICOUNT = 0
CALL SRAND(123.4567)
DO I = 1, 100
   X = RAND(0.0)
   IF ( (X > .5) .AND. (X < .6)) ICOUNT = ICOUNT + 1
END DO
WRITE (*,*) ICOUNT, 'Numbers between .5 and .6'
```
SSWRQQ

Returns the floating-point processor status word.

Prototype

USE IFLPORT
or INTERFACE
  SUBROUTINE SSWRQQ(STATUS)
    INTEGER(2) STATUS
  END SUBROUTINE
END INTERFACE

Usage

CALL SSWRQQ (STATUS)
STATUS INTEGER(2). Floating-point co-processor status word.
SSWRQQ performs the same function as GETSTATUSFPQQ and is provided for compatibility.

Example

USE IFLPORT
INTEGER(2) status
CALL SSWRQQ (status)
PRINT 10,STATUS
10 FORMAT('FP Status word was ',Z)
END
STAT

_Gets system information on a given file_

**Prototype**

**IA-32 systems**

INTERFACE

    INTEGER(4) FUNCTION STAT(NAME, STARRAY)
    CHARACTER(LEN=*) :: NAME
    INTEGER(4), DIMENSION(12) :: STARRAY
    END FUNCTION

    INTEGER(4) FUNCTION STATI8(NAME, STARRAY)
    CHARACTER(LEN=*) :: NAME
    INTEGER(8), DIMENSION(12) :: STARRAY
    END FUNCTION

END INTERFACE

**Itanium®-based systems**

INTERFACE STAT

    INTEGER(4) FUNCTION STATI8(NAME, STARRAY)
    CHARACTER(LEN=*) :: NAME
    INTEGER(8), DIMENSION(12) :: STARRAY
    END FUNCTION

END INTERFACE

**NAME**    a CHARACTER variable that specifies a pathname for the file that you want to check.

**STATARRAY** an integer array where system information about your file can be placed

**Description**

The filename must be currently connected to a logical unit, and must already exist when `STAT` is called.
This routine is thread-safe, and locks the associated stream before information is collected.

**NOTE.** On Windows, stat and lstat are equivalent. On Linux, if the file denoted by NAME is a link, lstat provides information on the link, while stat provides information on the file at the destination of the link.

**Output**

Errno set on failure, otherwise, STATARRAY filled in with information about the file. The contents of the STATARRAY on return are system dependent.

| stat (1) | Device that specifies the inode or handle (always 0 on Windows) |
| stat (2) | Inode or handle number (always 0 on Windows) |
| stat (3) | Protection level |
| stat (4) | Number of hard links to file (always 1 on Windows) |
| stat (5) | User ID of owner (always 1 on Windows) |
| stat (6) | Group ID of owner (always 1 on Windows) |
| stat (7) | Device type, if this inode is a device (always 0 on Windows) |
| stat (8) | Total size of file |
| stat (9) | File last access time (for Windows, only if file system non-FAT) |
| stat (10) | File last modify time (for Windows, same as stat(10)) |
| stat (11) | File last status change time (always 1 on Windows) |
| stat (12) | Optimal blocksize for file system I/O ops |
| stat (13) | Actual number of blocks allocated (Linux only, not present on Windows) |
SYSTEM

* Sends a command to the shell for execution*

**Prototype**

```fortran
INTERFACE
   INTEGER FUNCTION SYSTEM (COMMANDA)
   CHARACTER(LEN=*) COMMANDA
   END FUNCTION SYSTEM
END INTERFACE

COMMANDA     command to execute

**Description**

This function sends a command to the shell for execution as if it were typed on the command line.

**Output**

Exit status of the shell command.

SYSTEMQQ

* Executes a system command.*

**Prototype**

```fortran
USE IFLPORT

or

INTERFACE
   LOGICAL(4) FUNCTION SYSTEMQQ (COMMANDA)
   CHARACTER(LEN=*) COMMANDA
END INTERFACE
```
END FUNCTION SYSTEMQQ
END INTERFACE

Description
Executes a system command by passing a command string to the operating system's command interpreter.

Usage
result = SYSTEMQQ (COMMAND)
COMMANDA CHARACTER(LEN=*). Text of the command line to be passed to the operating system.

Results
The result is .TRUE. if successful; otherwise, .FALSE..

The SYSTEMQQ function lets you pass command shell commands as well as programs. SYSTEMQQ refers to the COMSPEC and PATH environment variables that locate the command interpreter file (usually named COMMAND.COM).

On Windows NT systems, the calling process waits until the command terminates. On Windows 95 and Windows 98 systems, the calling process does not currently wait in all cases; however, this may change in future implementations. To insure compatibility and consistent behavior, an image can be invoked directly by using the WIN32 API CreateProcess ( ) in your Fortran code.

If the function fails, you can call GETLASTERRORQQ to determine the reason. One of the following errors will be returned:

ERR$2BIG The argument list exceeds 128 bytes, or the space required for the environment formation exceeds 32K.

ERR$NOINT The command interpreter cannot be found.

ERR$NOEXEC The command interpreter file has an invalid format and is not executable.
ERR$NOMEM Not enough memory is available to execute the command; or the available memory has been corrupted; or an invalid block exists, indicating that the process making the call was not allocated properly.

The command line character limit for the SYSTEMQQ function is the same limit that your command shell accepts.

Example
USE IFLPORT
! Note: compile this example with /nbs
LOGICAL(4) RESULT
RESULT = SYSTEMQQ('dir "c:\program files"')
PRINT *, '*******'
PRINT *, 'Result returned from SYSTEMQQ was ',RESULT
END

TIME
Returns the current time

Prototype
INTERFACE
   SUBROUTINE TIME(STRING)
      CHARACTER(LEN=8) STRING
   END SUBROUTINE TIME
END INTERFACE

STRING Must be of type character and must provide at least 8 bytes of storage to contain the current time in the form: hh:mm:ss where hh is the current hour, mm the current minute, ss the number of seconds past the minute.
**Description**
This routine returns the system time in seconds since 00:00:00 GMT, January 1, 1970. It fills the STRING parameter with current time.

**Class**
Nonstandard subroutine.

**Output**
The elapsed time in seconds since 00:00:00 Greenwich Mean Time, January 1, 1970.

**Example**
The following code sets the character variable \texttt{tstr} to the current system time (for example, 16:20:07).

```fortran
CHARACTER(8) tstr
CALL TIME(tstr)
```

---

**TIMEF**

*Returns the elapsed time since last called*

**Prototype**

```fortran
INTERFACE
   REAL(4) FUNCTION TIMEF()
END FUNCTION TIMEF
END INTERFACE
```

TIME elapsed time in seconds
Description
This function returns the number of seconds that elapsed since the first time TIMEF was called, or zero if the called for the first time.

Output
The number of seconds that elapsed since the first time TIMEF was called, or zero if called for the first time.

TOPEN
Tape open

Prototype

INTERFACE

   INTEGER(4) FUNCTION TOPEN(LUNIT, DEVNAME, &
   LABELLED)

   INTEGER(4), INTENT(OUT) :: LUNIT
   CHARACTER(LEN=*) , INTENT(IN) :: DEVNAME
   LOGICAL(4), INTENT(IN) :: LABELLED

END FUNCTION TOPEN

END INTERFACE

TLU     A Fortran logical unit number of 0 to 99 range
DEVNAME the device name of the tape unit
LABELLED indicates whether the tape to be opened is a labelled tape: 1 for a labeled tape or zero for an unlabeled tape
Description

TOPEN opens a Fortran logical unit on a tape device for use with TREAD and WRITE.

NOTE. This function is for Win32 systems only.

Output

0 for successful open or most recent value of errno for an error.

TCLOSE

Close a tape file

Prototype

INTERFACE
   INTEGER(4) FUNCTION TCLOSE( TLU)
   INTEGER(4), INTENT(IN):: TLU
END FUNCTION TCLOSE
END INTERFACE

TLU a Win32 file handle descriptor

Description

Using a file handle obtained from a previous call to TOPEN, TCLOSE closes a tape file on a tape drive on your local system.

NOTE. This function is for Win32 systems only.
Output

TCLOSE returns zero if the close operation was successful, or returns errno if the close operation was not successful.

TREAD

Read from a tape file.

Prototype

INTERFACE

   INTEGER(4) FUNCTION TREAD(TLU, BUFFER)

   INTEGER(4), INTENT(IN) :: TLU
   CHARACTER(LEN=*) , INTENT(OUT) :: BUFFER

END FUNCTION TREAD

END INTERFACE

TLU is a file handle descriptor
BUFFER a character variable, array, or array section large enough to hold the next record on the tape

Description

TREAD reads the next logical record from an already opened tape file. You must have previously opened the file using the TOPEN routine, and obtained a valid Win32 system file handle.

NOTE. This function is for Win32 systems only.
**Output**

TREAD places the next logical record of a tape file in the input variable BUFFER. If the read operation is successful, TREAD returns zero as its result. If the read operation is not successful, TREAD returns `errno` as its result.

---

**TTYNAM**

*Checks if the unit is a terminal*

**Prototype**

```fortran
INTERFACE
  CHARACTER(LEN=*) FUNCTION TTYNAM(LUN)
  INTEGER(4) INTENT(IN) :: LUN
END FUNCTION TTYNAM
END INTERFACE
```

**Description**

This function determines whether a particular logical unit is connected to a terminal (TTY) display device.

**Output**

A string indicating the CHARACTER device name for a terminal device, or all blanks if not a terminal, or an error.
**TWRITE**

*Writes to a tape file*

**Prototype**

```fortran
INTERFACE
   INTEGER FUNCTION TWRITE(TLU, BUFFER)
   INTEGER(4), INTENT(IN) :: TLU
   CHARACTER(LEN=*) , INTENT(OUT) :: BUFFER
   END FUNCTION TREAD
END INTERFACE
```

**TLU** is a file handle descriptor

**BUFFER** a CHARACTER expression whose value is data to be written to a tape file on your local system

**Description**

*TWRITE* takes the data you pass to it in the input expression *BUFFER* and writes it as a logical record to a tape device on your local system. You must have previously opened the tape device with a call to *TOPEN*, and obtained a valid Win32 system file handle.

**Output**

*TWRITE* returns zero if the write operation was successful, and otherwise returns the system error code from *errno*.
UNLINK

Deletes a file by name

Prototype

INTERFACE
   INTEGER(4) FUNCTION UNLINK (NAME)
   CHARACTER(LEN=*) , INTENT(IN)::NAME
END FUNCTION UNLINK
END INTERFACE

NAME the name of the file you want to delete

Description

UNLINK deletes a file with the name specified by NAME. You must have adequate permission to delete the file. The name can be any character expression that results in a valid file name. The name can include a full path name, including drive letter.

Output

UNLINK returns a status code, which is zero if the deletion was successful, or the value of errno if the file deletion was not successful.

UNPACKTIMEQQ

Unpacks a packed time and date value.

Prototype

USE IFLPORT
or
IA-32 systems
INTERFACE
  SUBROUTINE UNPACKTIMEQQ(TIMEDATE, IYR, IMON, IDAY, &
                            IHR, IMIN, ISEC)
    INTEGER(4) TIMEDATE
    INTEGER(2) IYR, IMON, IDAY, IHR, IMIN, ISEC
  END SUBROUTINE
END INTERFACE

Itanium®-based systems
INTERFACE
SUBROUTINE UNPACKTIMEQQ (TIMEDATE, IYR, IMON, IDAY, &
                          IHR, IMIN, ISEC)
    INTEGER(8) TIMEDATE
    INTEGER(2) IYR, IMON, IDAY, IHR, IMIN, ISEC
  END SUBROUTINE
END INTERFACE

Description
Unpacks a packed time and date value into its component parts. See PACKTIMEQQ.

Usage
CALL UNPACKTIMEQQ(TIMEDATE, IYR, IMON, IDAY, IHR, IMIN, ISEC)
TIMEDATE   INTEGER(4). Packed time and date information.
IYR        INTEGER(2). Year (xxxx AD).
IMON       INTEGER(2). Month (1 - 12).
IDAY       INTEGER(2). Day (1 - 31).
IHR         INTEGER(2). Hour (0 - 23).
IMIN       INTEGER(2). Minute (0 - 59).
ISEC       INTEGER(2). Second (0 - 59).
GETFILEINFO returns time and date in a packed format. You can use UNPACKTIME to unpack these values. Use PACKTIME to repack times for passing to SETFILETIME. Packed times can be compared using relational operators.

Example

! Note, compile this example with /nbs.
USE IFLPORT
CHARACTER(80) file
TYPE (FILE$INFO) info
INTEGER(4) handle, result
INTEGER(2) iyr, imon, iday, ihr, imin, isec
file = 'd:\f90ps\bin\t???.*'
handle = FILE$FIRST
result = GETFILEINFO(file, info, handle)
CALL UNPACKTIME(info%lastwrite, iyr, imon, &
                 iday, ihr, imin, isec)
WRITE(*,*) iyr, imon, iday
WRITE(*,*) ihr, imin, isec
END

National Language Support Routines

National Language Support (NLS) procedures provide language localization and a subset of multi-byte character set (MBCS) NLS functions to let you write applications in different languages. To use an NLS routine, add the following statement to the program unit containing the procedure:

USE IFLPORT
or
INCLUDE
'<installation & directory>\...\include\iflport.f90'
Table 2-1 summarizes the NLS procedures. The names are listed in mixed case to make the mnemonics easier to understand. When writing your applications, you can use any case.

### Table 2-1 Multi-byte Routines and Functions Summary

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Locale Setting and Inquiry</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| NLSEnumCodepages
ptr => NLSEnumCodepages ( ) | Function              | Returns all the supported codepages on the system.                         |
| NLSEnumLocales
ptr => NLSEnumLocales ( )      | Function              | Returns all the languages and country combinations supported by the system. |
| NLSGetEnvironmentCodepage
result = NLSGetEnvironmentCodepage ( flags ) | Function              | Returns the codepage number for the system (Window) codepage or the console codepage. |
| NLSGetLocale
| NLSGetLocaleInfo
result = NLSGetLocaleInfo ( type, outstr ) | Function              | Returns requested information about the current local code set.           |
| NLSSetEnvironmentCodepage
result = NLSSetEnvironmentCodepage ( codepage, flags ) | Function              | Changes the codepage for the current console.                              |
| NLSSetLocale
result = NLSSetLocale ( language [, country] [, codepage ] ) | Function              | Sets the language, country, and codepage.                                  |

**continued**
## Table 2-1 Multi-byte Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Locale Formatting</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLSFormatCurrency</td>
<td>Function</td>
<td>Formats a number string and returns the correct currency string for the current locale.</td>
</tr>
<tr>
<td>result = NLSFormatCurrency( outstr, instr [, flags ] )</td>
<td>Function</td>
<td>Returns a correctly formatted string containing the date for the current locale.</td>
</tr>
<tr>
<td>NLSFormatNumber</td>
<td>Function</td>
<td>Formats a number string and returns the correct number string for the current locale.</td>
</tr>
<tr>
<td>result = NLSFormatNumber( outstr [, intime ] [, flags ] )</td>
<td>Function</td>
<td>Returns a correctly formatted string containing the time for the current locale.</td>
</tr>
<tr>
<td><strong>MBCS Inquiry</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBCharLen</td>
<td>Function</td>
<td>Returns the length, in bytes, of the first character in a multi-byte-character string.</td>
</tr>
<tr>
<td>result = MBCharLen( string)</td>
<td>Function</td>
<td>Returns the longest possible multi-byte character length, in bytes, for the current codepage.</td>
</tr>
<tr>
<td>MBCurMax</td>
<td>Function</td>
<td>Determines whether a given character is the lead (first) byte of a multi-byte character sequence.</td>
</tr>
<tr>
<td>result = MBCurMax( )</td>
<td>Function</td>
<td>Returns the number of characters in a multi-byte-character string, including trailing blanks.</td>
</tr>
<tr>
<td>MBLead</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>result = MBLead( char )</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>MBLen</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>result = MBLen( string )</td>
<td>Function</td>
<td></td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBLen_Trim</td>
<td>Function</td>
<td>Returns the number of characters in a multi-byte-character string, not including trailing blanks.</td>
</tr>
<tr>
<td>result = MBLen_Trim (string)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBNext</td>
<td>Function</td>
<td>Returns the position of the first lead byte or single-byte character immediately following the given position in a multi-byte-character string.</td>
</tr>
<tr>
<td>result = MBNext (string, position)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBPrev</td>
<td>Function</td>
<td>Returns the position of the first lead byte or single-byte character immediately preceding the given string position in a multi-byte-character string.</td>
</tr>
<tr>
<td>result = MBPrev (string, position)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBStrLead</td>
<td>Function</td>
<td>Performs a context-sensitive test to determine whether a given character byte in a string is a multi-byte-character lead byte.</td>
</tr>
<tr>
<td>result = MBStrLead (string, position)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBCS Conversion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBConvertMBToUnicode</td>
<td>Function</td>
<td>Converts a character string from a multi-byte codepage to a Unicode string.</td>
</tr>
<tr>
<td>result = MBConvertMBToUnicode (mbstr, unicodestr [, flags] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBConvertUnicodeToMB</td>
<td>Function</td>
<td>Converts a Unicode string to a multi-byte character string of the current codepage.</td>
</tr>
<tr>
<td>result = MBConvertUnicodeToMB (unicodestr, mbstr [,flags])</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2-1  Multi-byte Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MBCS Fortran Equivalent Procedures</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MBINCHARQQ</strong></td>
<td>Function</td>
<td>Same as INCHARQQ except that it can read a single multi-byte character at once and returns the number of bytes read.</td>
</tr>
<tr>
<td>result = MBINCHARQQ (string)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MBINDEX</strong></td>
<td>Function</td>
<td>Same as INDEX, except that multi-byte characters can be included in its arguments.</td>
</tr>
<tr>
<td>result = MBINDEX (string, substring [, back ] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MBLGE, MBLGT, MBLLE, MBLLT, MBLEQ, MBLNE</strong></td>
<td>Function</td>
<td>Same as LGE, LGT, LLE, and LLT, and the logical operators .EQ. and .NE., except that multi-byte characters can be included in their arguments. All these routines have the same arguments as shown for MBLGE.</td>
</tr>
<tr>
<td>result = MBLGE (string_a, string_b, [ flags ] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MBSCAN</strong></td>
<td>Function</td>
<td>Same as SCAN, except that multi-byte characters can be included in its arguments.</td>
</tr>
<tr>
<td>result = MBSCAN (string, set [, back ] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MBVERIFY</strong></td>
<td>Function</td>
<td>Same as VERIFY, except that multi-byte characters can be included in its arguments.</td>
</tr>
<tr>
<td>result = MBVERIFY (string, set [, back ] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MBJISTToJMS</strong></td>
<td>Function</td>
<td>Converts a Japan Industry Standard (JIS) character to a Microsoft Kanji (Shift JIS or JMS) character.</td>
</tr>
<tr>
<td><strong>MBJMSTToJIS</strong></td>
<td>Function</td>
<td>Converts a Microsoft Kanji (Shift JIS or JMS) character to a Japan Industry Standard (JIS) character.</td>
</tr>
</tbody>
</table>
Locale Setting and Inquiry Procedures

**NLSenumCodepages**

*Returns all the supported codepages on the system*

**Prototype**

```fortran
INTERFACE
  FUNCTION NLSenumCodepages()
    INTEGER(4), POINTER :: NLSenumCodepages (:)
  END FUNCTION
END INTERFACE
```

**Description**

Returns an array containing the code pages supported by the system, with each array element describing one valid codepage.

**NOTE.** *After use, the pointer returned by NLSenumCodepages should be deallocated with the DEALLOCATE statement.*

**Output**

Pointer to an array of codepages, with each element describing one supported codepage.
**NLSEnumLocales**

*Returns all the language and country combinations supported by the system*

**Prototype**

```plaintext
INTERFACE
  FUNCTION NLSEnumLocales()
    INTEGER(4), PARAMETER :: NLS$MaxLanguageLen = 64
    INTEGER(4), PARAMETER :: NLS$MaxCountryLen = 64
    TYPE NLS$EnumLocale
      SEQUENCE
        CHARACTER(LEN= NLS$MaxLanguageLen) Language
        CHARACTER(LEN= NLS$MaxCountryLen) Country
        INTEGER(4) DefaultWindowsCodepage
        INTEGER(4) DefaultConsoleCodepage
    END TYPE
    TYPE(NLS$EnumLocale), POINTER::NLSEnumLocales (:)
  END FUNCTION
END INTERFACE
```

**Description**

Returns an array containing the language and country combinations supported by the system, in which each array element describes one valid combination.

**NOTE.** *After use, the pointer returned by NLSEnumLocales should be deallocated with the DEALLOCATE statement.*
**Output**

Pointer to an array of locales, in which each array element describes one supported language and country combination.

If the application is a Windows or a QuickWin application, **NLS$DefaultWindowsCodepage** is the codepage used by default for the given language and country combination. If the application is a console application, **NLS$DefaultConsoleCodepage** is the codepage used by default for the given language and country combination.

---

**NLSGetEnvironmentCodepage**

*Returns the codepage number for the system or the console codepage*

---

**Prototype**

```fortran
INTERFACE
   INTEGER(4) FUNCTION NLSGetEnvironmentCodepage(FLAGS)
   INTEGER(4), INTENT(IN) :: FLAGS
END FUNCTION
END INTERFACE
```

**FLAGS**

Tells the function which codepage number to return.

Available values are:

- **NLS$ConsoleEnvironmentCodepage** - gets the codepage for the console
- **NLS$WindowsEnvironmentCodepage** - gets the current Windows codepage

**Description**

Returns the codepage number for the system (Window) codepage or the console codepage.
Output
Zero if successful; otherwise, returns one of the following error codes:
NLS$ErrorInvalidFlags
   indicates that FLAGS has an illegal value
NLS$ErrorNoConsole
   there is no console associated with the given application;
   therefore, operations with the console codepage are not possible.

NLSGetLocale

Returns the current language, country,
and/or codepage.

Prototype
INTERFACE
  SUBROUTINE NLSGetLocale (LANGUAGE, COUNTRY, &
   CODEPAGE)
    CHARACTER(LEN=*) , INTENT (OUT), OPTIONAL::LANGUAGE
    CHARACTER(LEN=*) , INTENT (OUT), OPTIONAL::COUNTRY
    INTEGER(4) , INTENT (OUT), OPTIONAL::CODEPAGE
  END SUBROUTINE
END INTERFACE
LANGUAGE       Optional, output. Character(LEN=*). Current language.
COUNTRY        Optional, output. Character(LEN=*). Current country.
CODEPAGE       Optional, output. Character(LEN=*). Current codepage.

Description
Retrieves the current language, country, and/or codepage.
NOTE. NLSGetLocale returns a valid codepage in codepage. It does not return one of the NLS$... symbolic constants that can be used with NLSSetLocale.

Output
Requested information about the current language, country, and codepage.

NLSGetLocaleInfo

Returns information about the current local code set

Prototype

INTERFACE
FUNCTION NLSGetLocaleInfo(INFOTYPE, &
OUTSTR)

INTEGER(4), INTENT(IN)::INFOTYPE
CHARACTER(LEN=*), INTENT(OUT)::OUTSTR

END FUNCTION

END INTERFACE

INFOTYPE Input NLS parameter requested. A list of parameter names is given in the NLS Locale Info Parameters in “Note.”

OUTSTR Output. Character(LEN=*). Parameter setting for the current locale. All parameter settings placed in OUTSTR are character strings, even numbers. If a parameter setting is numeric, the ASCII representation of the number is used. If the requested parameter is a
date or time string, an explanation of how to interpret the format in OUTSTR is given in NLS Date and Time Format.

**Description**

Returns requested information about the current local code set.

---

**NOTE.** The NLS$LI parameters are used for the argument INFOTYPE and select the locale information returned by NLSGetLocaleInfo in OUTSTR. You can perform an inclusive OR with NLS$NoUserOverride and any NLS$LI parameter. This causes NLSGetLocaleInfo to bypass any user overrides and always return the system default value.

---

Table 2-2 lists and describes all NLS$LI parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_ILANGUAGE</td>
<td>An ID indicating the language</td>
</tr>
<tr>
<td>NLS$LI_SLANGUAGE</td>
<td>The full localized name of the language.</td>
</tr>
<tr>
<td>NLS$LI_SENGLANGUAGE</td>
<td>The full English name of the language from the ISO Standard 639. This is limited to characters that map into the ASCII 127 character subset.</td>
</tr>
<tr>
<td>NLS$LI_SABBREVLANGNAME</td>
<td>The abbreviated name of the language, created by taking the 2-letter language abbreviation as found in ISO Standard 639 and adding a third letter as appropriate to indicate the sublanguage.</td>
</tr>
<tr>
<td>NLS$LI_SNATIVELANGNAME</td>
<td>The native name of the language.</td>
</tr>
</tbody>
</table>

continued
### Table 2-2 NLS$LI Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_ICOUNTRY</td>
<td>The country code, based on international phone codes, also referred to as IBM country codes.</td>
</tr>
<tr>
<td>NLS$LI_SCOUNTY</td>
<td>The full localized name of the country.</td>
</tr>
<tr>
<td>NLS$LI_SENGCOUNTRY</td>
<td>The full English name of the country. This will always be limited to characters that map into the ASCII 127 character subset.</td>
</tr>
<tr>
<td>NLS$LI_SABBREVCTRNAME</td>
<td>The abbreviated name of the country as per ISO Standard 3166.</td>
</tr>
<tr>
<td>NLS$LI_SNATIVECTRNAME</td>
<td>The native name of the country.</td>
</tr>
<tr>
<td>NLS$LI_IDEFAULTLANGUAGE</td>
<td>Language ID for the principal language spoken in this locale. This is provided so that partially specified locales can be completed with default values.</td>
</tr>
<tr>
<td>NLS$LI_IDEFAULTCOUNTRY</td>
<td>Country code for the principal country in this locale. This is provided so that partially specified locales can be completed with default values.</td>
</tr>
<tr>
<td>NLS$LI_IDEFAULTANSICODEPAGE</td>
<td>ANSI code page associated with this locale.</td>
</tr>
<tr>
<td>NLS$LI_IDEFAULTOEMCODEPAGE</td>
<td>OEM code page associated with the locale.</td>
</tr>
<tr>
<td>NLS$LI_SLIST</td>
<td>Character(s) used to separate list items, for example, comma in many locales.</td>
</tr>
<tr>
<td>NLS$LI_IMEASURE</td>
<td>This value is 0 if the metric system (S.I.) is used and 1 for the U.S. system of measurements.</td>
</tr>
</tbody>
</table>
### Table 2-2  NLS$LI Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_SDECIMAL</td>
<td>The character(s) used as decimal separator. This cannot be set to digits 0 - 9.</td>
</tr>
<tr>
<td>NLS$LI_STHOUSAND</td>
<td>The character(s) used as separator between groups of digits left of the decimal. This cannot be set to digits 0 - 9.</td>
</tr>
<tr>
<td>NLS$LI_SGROUPING</td>
<td>Sizes for each group of digits to the left of the decimal. An explicit size is needed for each group; sizes are separated by semicolons. If the last value is 0 the preceding value is repeated. To group thousands, specify “3;0”.</td>
</tr>
<tr>
<td>NLS$LI_IDIGITS</td>
<td>The number of decimal digits.</td>
</tr>
</tbody>
</table>
| NLS$LI_IDIGITSNLS$LI_ILZERO | Determines whether to use leading zeros in decimal fields:  
0 - Use no leading zeros  
1 - Use leading zeros.                                                                            |
| NLS$LI_INEGNUMBER  | Determines how negative numbers are represented:  
0 - Puts negative numbers in parentheses: (1.1)  
1 - Puts a minus sign in front: -1.1  
2 - Puts a minus sign followed by a space in front: - 1.1  
3 - Puts a minus sign after: 1.1-  
4 - Puts a space then a minus sign after: 1.1- |

continued
Table 2-2  NLS$LI Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_SNATIVEDEGDIGITS</td>
<td>The ten characters that are the native equivalent to the ASCII 0-9.</td>
</tr>
<tr>
<td>NLS$LI_SCURRENCY</td>
<td>The string used as the local monetary symbol. Cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$LI_SINTLSYMBOL</td>
<td>Three characters of the International monetary symbol specified in ISO 4217 “Codes for the Representation of Currencies and Funds”, followed by the character separating this string from the amount.</td>
</tr>
<tr>
<td>NLS$LI_SMONDECIMALSEP</td>
<td>The character(s) used as monetary decimal separator. This cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$LI_SMONTHOUSANDSEP</td>
<td>The character(s) used as monetary separator between groups of digits left of the decimal. Cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$LI_SMONGROUPING</td>
<td>Sizes for each group of monetary digits to the left of the decimal. If the last value is 0, the preceding value is repeated. To group thousands, specify “3;0”.</td>
</tr>
<tr>
<td>NLS$LI_ICURRDIGITS</td>
<td>Number of decimal digits for the local monetary format.</td>
</tr>
<tr>
<td>NLS$LI_IINTLCURRDIGITS</td>
<td>Number of decimal digits for the international monetary format.</td>
</tr>
</tbody>
</table>

continued
### Table 2-2  NLS$LI Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_ICURRENCY</td>
<td>Determines how positive currency is represented:</td>
</tr>
<tr>
<td></td>
<td>0 - Puts currency symbol in front with no separation: $1.1</td>
</tr>
<tr>
<td></td>
<td>1 - Puts currency symbol in back with no separation: 1.1$</td>
</tr>
<tr>
<td></td>
<td>2 - Puts currency symbol in front with single space after: $ 1.1</td>
</tr>
<tr>
<td></td>
<td>3 - Puts currency symbol in back with single space before: 1.1 $</td>
</tr>
<tr>
<td>NLS$LI_INEGCURREN</td>
<td>Determines how negative currency is represented:</td>
</tr>
<tr>
<td></td>
<td>0: ($1.1)</td>
</tr>
<tr>
<td></td>
<td>1: -$1.1</td>
</tr>
<tr>
<td></td>
<td>2: $-1.1</td>
</tr>
<tr>
<td></td>
<td>3: $1.1-</td>
</tr>
<tr>
<td></td>
<td>4: (1.1$)</td>
</tr>
<tr>
<td></td>
<td>5: -1.1$</td>
</tr>
<tr>
<td></td>
<td>6: 1.1-$</td>
</tr>
<tr>
<td></td>
<td>7: 1.1$-</td>
</tr>
<tr>
<td></td>
<td>8: -1.1 $ (space before $)</td>
</tr>
<tr>
<td></td>
<td>9: -$ 1.1 (space after $)</td>
</tr>
<tr>
<td></td>
<td>10: 1.1 $- (space before $)</td>
</tr>
<tr>
<td></td>
<td>11: $ 1.1- (space after $)</td>
</tr>
<tr>
<td></td>
<td>12: $ -1.1 (space after $)</td>
</tr>
<tr>
<td></td>
<td>13: 1.1- $ (space before $)</td>
</tr>
<tr>
<td></td>
<td>14: ($ 1.1) (space after $)</td>
</tr>
<tr>
<td></td>
<td>15: (1.1 $) (space before $)</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLS$LI_SPOSITIVESIGN</td>
<td>String value for the positive sign. Cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$LI_SNEGATIVESIGN</td>
<td>String value for the negative sign. Cannot be set to digits 0-9.</td>
</tr>
<tr>
<td>NLS$LI_IPOSSIGNPOSN</td>
<td>Determines the formatting index for positive values:</td>
</tr>
<tr>
<td></td>
<td>0 - Parenthesis surround the amount and the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>1 - The sign string precedes the amount and the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>2 - The sign string follows the amount and the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>3 - The sign string immediately precedes the monetary symbol</td>
</tr>
<tr>
<td></td>
<td>4 - The sign string immediately follows the monetary symbol</td>
</tr>
<tr>
<td>NLS$LI_INEGSIGNPOSN</td>
<td>Determines the formatting index for negative values. Same values as for NLS$LI_IPOSSIGNPOSN</td>
</tr>
<tr>
<td>NLS$LI_IPOSSYMPRECEDES</td>
<td>1 if the monetary symbol precedes, 0 if it follows a positive amount.</td>
</tr>
<tr>
<td>NLS$LI_IPOSSEPBYSPACE</td>
<td>1 if the monetary symbol is separated by a space from a positive amount, 0 otherwise.</td>
</tr>
<tr>
<td>NLS$LI_INEGSYMPRECEDES</td>
<td>1 if the monetary symbol precedes, 0 if it follows a negative amount.</td>
</tr>
<tr>
<td>NLS$LI_INEGSEPBYSPACE</td>
<td>1 if the monetary symbol is separated by a space from a negative amount, 0 otherwise.</td>
</tr>
</tbody>
</table>
Table 2-2  **NLS$LI Parameters** (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NLS$LI_STIMEFORMAT</strong></td>
<td>Time formatting string. See the NLS Date and Time Format section for explanations of the valid strings.</td>
</tr>
<tr>
<td><strong>NLS$LI_STIME</strong></td>
<td>Character(s) for the time separator. Cannot be set to digits 0-9.</td>
</tr>
</tbody>
</table>
| **NLS$LI_ITIME**       | Time format:  
                          0 - Use 12-hour format  
                          1 - Use 24-hour format |
| **NLS$LI_ITLZERO**     | Determines whether to use leading zeros in time fields:  
                          0 - Use no leading zeros  
                          1 - Use leading zeros for hours |
| **NLS$LI_S1159**       | String for the AM designator                                              |
| **NLS$LI_S2359**       | String for the PM designator                                              |
| **NLS$LI_SSHORTDATE**  | Short Date formatting string for this locale. The d, M and y should have the day, month, and year substituted, respectively. See the NLS Date and Time Format section for explanations of the valid strings. |
| **NLS$LI_SDATE**       | Character(s) for the date separator. Cannot be set to digits 0-9.           |
| **NLS$LI_IDATE**       | Short Date format ordering:  
                          0 - Month-Day-Year  
                          1 - Day-Month-Year  
                          2 - Year-Month-Day |

continued
### Table 2-2  NLS$LI Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_ICENTURY</td>
<td>Specifies whether to use full 4-digit century for the short date only:</td>
</tr>
<tr>
<td></td>
<td>0 - Two-digit year</td>
</tr>
<tr>
<td></td>
<td>1 - Full century</td>
</tr>
<tr>
<td>NLS$LI_IDAYLZERO</td>
<td>Specifies whether to use leading zeros in day fields for the short date only:</td>
</tr>
<tr>
<td></td>
<td>0 - Use no leading zeros</td>
</tr>
<tr>
<td></td>
<td>1 - Use leading zeros</td>
</tr>
<tr>
<td>NLS$LI_IMONLZERO</td>
<td>Specifies whether to use leading zeros in month fields for the short date only:</td>
</tr>
<tr>
<td></td>
<td>0 - Use no leading zeros</td>
</tr>
<tr>
<td></td>
<td>1 - Use leading zeros</td>
</tr>
<tr>
<td>NLS$LI_SLONGDATE</td>
<td>Long Date formatting string for this locale.</td>
</tr>
<tr>
<td></td>
<td>The string returned may contain a string within single quotes (&quot;'). Any characters within single quotes should be left as is. The d, M and y should have the day, month, and year substituted, respectively.</td>
</tr>
<tr>
<td>NLS$LI_ILDATE</td>
<td>Long Date format ordering:</td>
</tr>
<tr>
<td></td>
<td>0 - Month-Day-Year</td>
</tr>
<tr>
<td></td>
<td>1 - Day-Month-Year</td>
</tr>
<tr>
<td></td>
<td>2 - Year-Month-Day</td>
</tr>
<tr>
<td>NLS$LI_ICALENDARTYPE</td>
<td>Specifies which type of calendar is currently being used:</td>
</tr>
<tr>
<td></td>
<td>1 - Gregorian (as in United States)</td>
</tr>
<tr>
<td></td>
<td>2 - Gregorian (English strings always)</td>
</tr>
<tr>
<td></td>
<td>3 - Era: Year of the Emperor (Japan)</td>
</tr>
<tr>
<td></td>
<td>4 - Era: Year of the Republic of China</td>
</tr>
<tr>
<td></td>
<td>5 - Tangun Era (Korea)</td>
</tr>
</tbody>
</table>

continued
### Table 2-2 NLS$LI Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NLS$LILOPTONALCALENDAR</strong></td>
<td>Specifies which additional calendar types are valid and available for this locale. This can be a null separated list of all valid optional calendars: 0 - No additional types valid 1 - Gregorian (localized) 2 - Gregorian (English strings always) 3 - Era: Year of the Emperor (Japan) 4 - Era: Year of the Republic of China 5 - Tungun Era (Korea)</td>
</tr>
<tr>
<td><strong>NLS$LI_IFIRSTDAYOFWEEK</strong></td>
<td>Specifies which day is considered first in a week: 0 - SDAYNAME1 1 - SDAYNAME2 2 - SDAYNAME3 3 - SDAYNAME4 4 - SDAYNAME5 5 - SDAYNAME6 6 - SDAYNAME7</td>
</tr>
<tr>
<td><strong>NLS$LI_IFIRSTWEEKOFYEAR</strong></td>
<td>Specifies which week of the year is considered first: 0 - Week containing 1/1 1 - First full week following 1/1 2 - First week containing at least 4 days</td>
</tr>
<tr>
<td><strong>NLS$LI_SDAYNAME1</strong></td>
<td>Long name for Monday</td>
</tr>
<tr>
<td><strong>NLS$LI_SDAYNAME2</strong></td>
<td>Long name for Tuesday</td>
</tr>
<tr>
<td><strong>NLS$LI_SDAYNAME3</strong></td>
<td>Long name for Wednesday</td>
</tr>
<tr>
<td><strong>NLS$LI_SDAYNAME4</strong></td>
<td>Long name for Thursday</td>
</tr>
<tr>
<td><strong>NLS$LI_SDAYNAME5</strong></td>
<td>Long name for Friday</td>
</tr>
<tr>
<td><strong>NLS$LI_SDAYNAME6</strong></td>
<td>Long name for Saturday</td>
</tr>
<tr>
<td><strong>NLS$LI_SDAYNAME7</strong></td>
<td>Long name for Sunday</td>
</tr>
<tr>
<td><strong>NLS$LI_SABBREVDAYNAME1</strong></td>
<td>Abbreviated name for Monday</td>
</tr>
<tr>
<td><strong>NLS$LI_SABBREVDAYNAME2</strong></td>
<td>Abbreviated name for Tuesday</td>
</tr>
<tr>
<td><strong>NLS$LI_SABBREVDAYNAME3</strong></td>
<td>Abbreviated name for Wednesday</td>
</tr>
</tbody>
</table>
Table 2-2  NLS$LI Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS$LI_SABBREVDAYNAME4</td>
<td>Abbreviated name for Thursday</td>
</tr>
<tr>
<td>NLS$LI_SABBREVDAYNAME5</td>
<td>Abbreviated name for Friday</td>
</tr>
<tr>
<td>NLS$LI_SABBREVDAYNAME6</td>
<td>Abbreviated name for Saturday</td>
</tr>
<tr>
<td>NLS$LI_SABBREVDAYNAME7</td>
<td>Abbreviated name for Sunday</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME1</td>
<td>Long name for January</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME2</td>
<td>Long name for February</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME3</td>
<td>Long name for March</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME4</td>
<td>Long name for April</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME5</td>
<td>Long name for May</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME6</td>
<td>Long name for June</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME7</td>
<td>Long name for July</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME8</td>
<td>Long name for August</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME9</td>
<td>Long name for September</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME10</td>
<td>Long name for October</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME11</td>
<td>Long name for November</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME12</td>
<td>Long name for December</td>
</tr>
<tr>
<td>NLS$LI_SMONTHNAME13</td>
<td>Long name for 13th month (if exists)</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME1</td>
<td>Abbreviated name for January</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME2</td>
<td>Abbreviated name for February</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME3</td>
<td>Abbreviated name for March</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME4</td>
<td>Abbreviated name for April</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME5</td>
<td>Abbreviated name for May</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME6</td>
<td>Abbreviated name for June</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME7</td>
<td>Abbreviated name for July</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME8</td>
<td>Abbreviated name for August</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME9</td>
<td>Abbreviated name for September</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME10</td>
<td>Abbreviated name for October</td>
</tr>
<tr>
<td>NLS$LI_SABBREVMONTHNAME11</td>
<td>Abbreviated name for November</td>
</tr>
</tbody>
</table>


### Output

Returns the number of characters written to `OUTSTR` if successful. If `OUTSTR` has 0 length, the number of characters required to hold the requested information is returned. Otherwise, one of the following error codes returns:

- **NLS$ErrorInvalidLIType**
  
  The given `INFOTYPE` is invalid.

- **NLS$ErrorInsufficientBuffer**
  
  The `OUTSTR` buffer was too small, but was not 0 (so that the needed size would be returned).

### NLSSetEnvironmentCodepage

*Changes the codepage for the current console*

#### Prototype

```plaintext
INTERFACE
    INTEGER(4) FUNCTION NLSSetEnvironmentCodepage(CODEPAGE, FLAGS)
    INTEGER(4), INTENT(IN) :: CODEPAGE
    INTEGER(4), INTENT(IN) :: FLAGS
END FUNCTION
END INTERFACE
```
Description
Sets the codepage for the current console. The specified codepage affects the current console program and any other programs launched from the same console. It does not affect other open consoles or any consoles opened later.

NOTE. The FLAGS argument must be NLS$ConsoleEnvironmentCodepage; it cannot be NLS$WindowsEnvironmentCodepage. NLS$SetEnvironmentCodepage does not affect the Windows codepage

Output
Returns zero if successful. Otherwise, returns one of the following error codes:
NLS$ErrorInvalidCodepage
CODEPAGE is invalid or not installed on the system
NLS$ErrorInvalidFlags
FLAGS is not valid.
NLS$ErrorNoConsole
There is no console associated with the given application; therefore operations, with the console codepage are not possible.
**NLSSetLocale**

*Sets the language, country, and codepage*

**Prototype**

```plaintext
INTERFACE

INTEGER(4) FUNCTION NLSSetLocale(LANGUAGE, COUNTRY, & CODEPAGE)

  CHARACTER(LEN=*)   :: LANGUAGE
  CHARACTER(LEN=*)   :: COUNTRY
  INTEGER(4)        :: CODEPAGE

END FUNCTION

END INTERFACE
```

**LANGUAGE**

Input. CHARACTER (LEN=*). One of the languages supported by the Win32* NLS APIs.

**COUNTRY**

Optional, input. CHARACTER (LEN=*). If specified, characterizes the language further. If omitted, the default country for the language is set.

**CODEPAGE**

Optional, input. INTEGER (4). If specified, codepage to use for all character-oriented NLS functions. Can be any valid supported codepage or one of the following predefined values:

- **NLS$CurrentCodepage**
  
The codepage is not changed. Only the language and country settings are altered by the function.

- **NLS$ConsoleEnvironmentCodepage**
  
The codepage is changed to the default environment codepage currently in effect for console programs.
NLS$ConsoleLanguageCodepage
The codepage is changed to the default console codepage for the language and country combination specified.

NLS$WindowsEnvironmentCodepage
The codepage is changed to the default environment codepage currently in effect for Windows programs.

NLS$WindowsLanguageCodepage
The codepage is changed to the default Windows codepage for the language and country combination specified.

If you omit CODEPAGE, it defaults to NLS$WindowsLanguageCodepage. At program startup, NLS$WindowsEnvironmentCodepage is used to set the codepage.

**Description**
Sets the current language, country, and/or codepage.

**NOTE.** NLSSetLocale works on installed locales only. Windows NT and Windows 95 support many locales, but these must be installed through the system Windows NT Control Panel/International menu or the Windows 95 Control Panel/Regional Settings menu.

In addition to the note above take into consideration the following:

- When doing mixed-language programming with Fortran and C, calling NLSSetLocale with a codepage other than the default environment Windows codepage causes the codepage in the C run-time library to change by calling C language setmbcp( ) routine with the new codepage. Conversely, changing the C run-time library codepage does not change the codepage in the Fortran NLS library.
- Calling NLSSetLocale has no effect on the locale used by C programs. The locale set with C language setlocale( ) routine is independent of NLSSetLocale.
- Calling NLSSetLocale with the default environment console codepage, NLS$ConsoleEnvironmentCodepage, causes an implicit call to the Win32 API SetFileApisToOEM(). Calling NLSSetLocale with any other codepage causes a call to SetFileApisToANSI().

**Output**

Zero if successful. Otherwise, one of the following error codes may be returned:

- NLS$ErrorInvalidLanguage
  LANGUAGE is invalid or not supported.

- NLS$ErrorInvalidCountry
  COUNTRY is invalid or is not valid with the language specified.

- NLS$ErrorInvalidCodepage
  CODEPAGE is invalid or not installed on the system.

**Locale Formatting Procedures**

**NLSFormatCurrency**

*Returns a correctly formatted currency string for the current locale*

**Prototype**

```fortran
INTERFACE
   INTEGER(4) FUNCTION NLSFormatCurrency(OUTSTR, &
                                       INSTR, FLAGS)
   INTEGER(4), INTENT(IN), OPTIONAL :: FLAGS
   CHARACTER(LEN=*) , INTENT(IN) :: INSTR
   CHARACTER(LEN=*) , INTENT(OUT) :: OUTSTR
END INTERFACE
```
END FUNCTION
END INTERFACE

OUTSTR  Output. CHARACTER(LEN=*)  String containing the correctly formatted currency for the current locale. If OUTSTR is longer than the formatted currency, it is blank-padded.

INSTR   Input. CHARACTER(LEN=*)  Number string to be formatted. Can contain only the characters 0' through 9', one decimal point (a period) if a floating-point value, and a minus sign in the first position if negative. All other characters are invalid and cause the function to return an error.

FLAGS   Optional, input. INTEGER(4)  If specified, modifies the currency conversion. If you omit FLAGS s, the flag NLS$Normal is used. Available values are:
         NLS$Normal
               No special formatting
         NLS$NoUserOverride
               Do not use user overrides

Description
Formats a number string and returns the correct currency string for the current locale.

Output
Number of characters written to OUTSTR (bytes are counted, not multibyte characters), or one of the following negative values if an error occurs:

NLS$ErrorInsufficientBuffer
       OUTSTR buffer is too small

NLS$ErrorInvalidInput
       FLAGS has an illegal value

NLS$ErrorInvalidFlags
       INSTR has an illegal value.
NLSFormatDate

Returns a correctly formatted string containing the date for the current locale

Prototype

INTERFACE

  INTEGER(4) FUNCTION NLSFormatDate(OUTSTR, INTIME, &
    FLAGS)
  INTEGER(4), INTENT(IN), OPTIONAL :: FLAGS, INTIME
  CHARACTER(LEN=*) , INTENT(OUT) :: OUTSTR
END FUNCTION
END INTERFACE

OUTSTR Output. CHARACTER(LEN=*) . String containing the correctly formatted date for the current locale. If OUTSTR is longer than the formatted date, it is blank-padded.

INTIME Optional, input. INTEGER(4) . If specified, date to be formatted for the current locale. Must be an integer date such as the packed time created with PACKTIMEQQ. If you omit INTIME, the current system date is formatted and returned in OUTSTR.

FLAGS Optional, input. INTEGER(4) . If specified, modifies the date conversion. If you omit FLAGS, the flag NLS$Normal is used. Available values are:

  NLS$Normal No special formatting
  NLS$NoUserOverride Do not use user overrides
  NLS$UseAltCalendar Use the locale's alternate calendar
NLS$LongDate
Use local long date format

NLS$ShortDate
Use local short date format

**Description**
Returns a correctly formatted string containing the date for the current locale.

**Output**
Number of characters written to OUTSTR (bytes are counted, not multibyte characters), or one of the following negative values if an error occurs:

NLS$ErrorInsufficientBuffer
OUTSTR buffer is too small

NLS$ErrorInvalidInput
INTIME has an illegal value

NLS$ErrorInvalidFlags
FLAGS has an illegal value

---

**NLSFormatNumber**

*Returns a correctly formatted number string for the current locale*

**Prototype**

```
INTERFACE
   INTEGER(4) FUNCTION NLSFormatNumber(OUTSTR, INSTR,&
   FLAGS)
   INTEGER(4), INTENT(IN), OPTIONAL :: FLAGS
   CHARACTER(LEN=*), INTENT(IN) :: INSTR
```
FUNCTION
CHARACTER(LEN=*) , INTENT(OUT) :: OUTSTR
END FUNCTION

END INTERFACE

OUTSTR
Output. CHARACTER(LEN=*) String containing the correctly formatted number for the current locale. If OUTSTR is longer than the formatted number, it is blank-padded.

INSTR
Input. CHARACTER(LEN=*) Number string to be formatted. Can only contain the characters 0 through 9, one decimal point (a period) if a floating-point value, and a minus sign in the first position if negative. All other characters are invalid and cause the function to return an error.

FLAGS
Optional, input. INTEGER(4). If specified, modifies the number conversion. If you omit FLAGS, the flag NLS$Normal is used. Available values are:

- NLS$Normal  No special formatting
- NLS$NoUserOverride  Do not use user overrides

Description
Formats a number string and returns the correct number string for the current locale.

Output
Number of characters written to OUTSTR (bytes are counted, not multibyte characters), or one of the following negative values if an error occurs:

- NLS$ErrorInsufficientBuffer  OUTSTR buffer is too small
- NLS$ErrorInvalidInput  INSTR has an illegal value
NLS$ErrorInvalidFlags

FLAGS has an illegal value

NLSFormatTime

Returns a correctly formatted string containing the time for the current locale

Prototype

INTERFACE

INTEGER(4) FUNCTION NLSFormatTime(OUTSTR, INTIME, &
FLAGS)

INTEGER(4), INTENT(IN), OPTIONAL :: FLAGS, INTIME
CHARACTER(LEN=*) , INTENT(OUT) :: OUTSTR
END FUNCTION
END INTERFACE

OUTSTR Output. CHARACTER(LEN=*). String containing the correctly formatted time for the current locale. If OUTSTR is longer than the formatted time, it is blank-padded.

INTIME Optional, input. INTEGER(4). If specified, time to be formatted for the current locale. Must be an integer time such as the packed time created with PACKTIMEQQ. If you omit INTIME, the current system time is formatted and returned in OUTSTR.

FLAGS Optional, input. INTEGER(4). If specified, modifies the time conversion. If you omit FLAGS, the flag NLS$Normal is used. Available values are:

NLS$Normal No special formatting
NLS$NoUserOverride Do not use user overrides
**NLS$NoMinutesOrSeconds**
Do not return minutes or seconds

**NLS$NoSeconds**
Do not return seconds

**NLS$NoTimeMarker**
Do not add a time marker string

**NLS$Force24HourFormat**
Return string in 24 hour format

**Description**
Returns a correctly formatted string containing the time for the current locale.

**Output**
Number of characters written to OUTSTR (bytes are counted, not multibyte characters), or one of the following negative values if an error occurs:

**NLS$ErrorInsufficientBuffer**
OUTSTR buffer is too small

**NLS$ErrorInvalidInput**
INTIME has an illegal value

**NLS$ErrorInvalidFlags**
FLAGS has an illegal value
MBCS Inquiry Procedures

MBCharLen

Returns the length, in bytes, of the first character in a multibyte-character string.

Prototype

```fortran
INTERFACE
    INTEGER(4) FUNCTION MBCharLen(STRING)
    CHARACTER(LEN=*) , INTENT(IN) :: STRING
END FUNCTION
END INTERFACE
```

STRING

Input. CHARACTER(LEN=*) . String containing the character whose length is to be determined. Can contain multibyte characters.

Description

Returns the length, in bytes, of the first character in a multibyte-character string.

NOTE. MBCharLen does not test for multibyte character validity.

Output

Number of bytes in the first character contained in string. Returns 0 if STRING has no characters (is length 0).
MBCurMax

*Returns the longest possible multibyte character length, in bytes, for the current codepage*

**Prototype**

```fortran
INTERFACE
   INTEGER(4) FUNCTION MBCurMax()
   END FUNCTION
END INTERFACE
```

**Description**

Returns the longest possible multibyte character length, in bytes, for the current codepage.

**NOTE.** The `MBLenMax` parameter, defined in the module `IFLPORT.F90`, is the longest length, in bytes, of any character in any codepage installed on the system.

**Output**

Longest possible multibyte character, in bytes, for the current codepage.
MBLen

Returns the number of characters in a multibyte-character string, including trailing blanks

Prototype

INTERFACE
  INTEGER(4) FUNCTION MBLen(STRING)
  CHARACTER(LEN=*) , INTENT(IN) :: STRING
END FUNCTION
END INTERFACE
STRING Input. CHARACTER(LEN=*). String whose characters are to be counted. Can contain multibyte characters.

Description

Returns the number of characters in a multibyte-character string, including trailing blanks.

NOTE. MBLen recognizes multibyte-character sequences according to the multibyte codepage currently in use. It does not test for multibyte-character validity

Output

Number of characters in STRING.
**MBLen_Trim**

*Returns the number of characters in a multibyte-character string, not including trailing blanks*

**Prototype**

```
INTERFACE
  INTEGER(4) FUNCTION MBLen_Trim(STRING)
  CHARACTER(LEN=*) :: STRING
END FUNCTION
END INTERFACE
```

**STRING**

Input. CHARACTER(LEN=*). String whose characters are to be counted. Can contain multibyte characters.

**Description**

Returns the number of characters in a multibyte-character string, not including trailing blanks.

**NOTE.** MBLen_Trim recognizes multibyte-character sequences according to the multibyte codepage currently in use. It does not test for multibyte-character validity.

**Output**

Number of characters in STRING minus any trailing blanks (blanks are bytes containing character 32 (hex 20) in the ASCII collating sequence).
**MBNext**

*Returns the position of the first lead byte or single-byte character immediately following the given position in a multibyte-character string*

---

**Prototype**

```fortran
INTERFACE
  INTEGER(4) FUNCTION MBNext (STRING, POSITION)
  CHARACTER (LEN=*) , INTENT(IN) :: STRING
  INTEGER(4) , INTENT(IN) :: POSITION
END FUNCTION
END INTERFACE
```

**STRING**

Input. CHARACTER (LEN=*) String to be searched for the first lead byte or single-byte character after the current position. Can contain multibyte characters.

**POSITION**

Input. INTEGER (4). Position in STRING to search from. Must be the position of a lead byte or a single-byte character. Cannot be the position of a trail (second) byte of a multibyte character.

**Description**

Returns the position of the first lead byte or single-byte character immediately following the given position in a multibyte-character string.

**Output**

Position of the first lead byte or single-byte character in STRING immediately following the position given in POSITION, or 0 if no following first byte is found in STRING.
**MBPrev**

*Returns the position of the first lead byte or single-byte character immediately preceding the given string position in a multibyte-character string*

---

**Prototype**

INTERFACE

   INTEGER(4) FUNCTION MBPrev(STRING, POSITION)
   CHARACTER(LEN=*) , INTENT(IN) :: STRING
   INTEGER(4) , INTENT(IN) :: POSITION

END FUNCTION
END INTERFACE

STRING

   Input. CHARACTER(LEN=*). String to be searched for the first lead byte or single-byte character before the current position. Can contain multibyte characters.

POSITION

   Input. INTEGER(4). Position in STRING to search from. Must be the position of a lead byte or single-byte character. Cannot be the position of the trail (second) byte of a multibyte character.

**Description**

Returns the position of the first lead byte or single-byte character immediately preceding the given string position in a multibyte-character string.

**Input**

Position of the first lead byte or single-byte character in STRING immediately preceding the position given in POSITION, or 0 if no preceding first byte is found in STRING.
MBStrLead

Performs a context-sensitive test to determine whether a given character byte in a string is a multibyte-character lead byte

Prototype

INTERFACE

LOGICAL(4) FUNCTION MBStrLead(STRING, POSITION)
CHARACTER(LEN=*) , INTENT(IN) :: STRING
INTEGER(4) , INTENT(IN) :: POSITION
END FUNCTION

END INTERFACE

STRING Input. CHARACTER(LEN=*) . String containing the character byte to be tested for lead status.

POSITION Input. INTEGER(4) . Position in STRING of the character byte in the string to be tested.

Description

Performs a context-sensitive test to determine whether a given character byte in a string is a multibyte-character lead byte.

NOTE. MBStrLead is passed a whole string and can identify any byte within the string as a lead or trail byte because it performs a context-sensitive test, scanning all the way back to the beginning of a string if necessary to establish context. MBLead is passed only one character at a time and must start on a lead byte and step through a string one character at a time to establish context for the character. Thus, MBStrLead can be much slower than MBLead (up to n times slower, where n is the length of the string).
Output

Returns .TRUE. if the character byte in POSITION of STRING is a lead byte; otherwise, .FALSE..

MBCS Conversion Procedures

**MBConvertMBToUnicode**

*Converts a character string from a multi-byte codepage to a Unicode string*

**Prototype**

```plaintext
INTERFACE
   INTEGER(4) FUNCTION MBConvertMBToUnicode(MBSTR, &
      UNICODESTR, FLAGS)
   CHARACTER(LEN=*) , INTENT(IN) :: MBSTR
   INTEGER(2) , DIMENSION(:) , INTENT(OUT) :: UNICODESTR
   INTEGER(4) , INTENT(IN) , OPTIONAL :: FLAGS
END FUNCTION
END INTERFACE
```

**MBSTR**

Input. CHARACTER(LEN=*). Multibyte codepage string to be converted.

**UNICODESTR**

Output. INTEGER(2). Array of integers that is the translation of the input string into Unicode.

**FLAGS**

Optional, input. INTEGER(4). If specified, modifies the string conversion. If FLAGS is omitted, the value NLS$Precomposed is used. Available values are:

- NLS$Precomposed
  - Use precomposed characters always. (default)
NLS$Composite
Use composite wide characters always.

NLS$UseGlyphChars
Use glyph characters instead of control characters.

NLS$ErrorOnInvalidChars
Returns -1 if an invalid input character is encountered.

The flags NLS$Precomposed and NLS$Composite are mutually exclusive. You can combine NLS$UseGlyphChars with either NLS$Precomposed or NLS$Composite using an inclusive OR (IOR or OR).

**Description**
Converts a multibyte-character string from the current codepage to a Unicode string.

**NOTE.** By default, or if FLAGS is set to NLS$Precomposed, the function MBConvertMBToUnicode attempts to translate the multibyte codepage string to a precomposed Unicode string. If a precomposed form does not exist, the function attempts to translate the codepage string to a composite form.

**Output**
If no error occurs, returns the number of bytes written to UNICODESTR (bytes are counted, not characters), or the number of bytes required to hold the output string if UNICODESTR has zero size. If the UNICODESTR array is bigger than needed to hold the translation, the extra elements are set to 0. If UNICODESTR has zero size, the function returns the number of bytes required to hold the translation and nothing is written to UNICODESTR.
If an error occurs, one of the following negative values is returned:

**NLS$ErrorInsufficientBuffer**
The **UNICODESTR** argument is too small, but not zero size so that the needed number of bytes would be returned.

**NLS$ErrorInvalidFlags**
The **FLAGS** argument has an illegal value.

**NLS$ErrorInvalidCharacter**
A character with no Unicode translation was encountered in **MBSTR**. This error can occur only if the **NLS$InvalidCharsError** flag was used in **FLAGS**.

---

**MBConvertUnicodeToMB**

*Converts a Unicode string to a multi-byte character string of the current codepage*

---

**Prototype**

```fortran
INTERFACE
  INTEGER(4) FUNCTION MBConvertUnicodeToMB(UNICODESTR, MBSTR, FLAGS)
  INTEGER(2), DIMENSION(:), INTENT(IN)::UNICODESTR
  CHARACTER(LEN=*) , INTENT(OUT)::MBSTR
  INTEGER(4), OPTIONAL, INTENT(IN)::FLAGS
END FUNCTION
END INTERFACE

UNICODESTR Input. INTEGER(2). Array of integers holding the Unicode string to be translated.
MBSTR  Output. CHARACTER (LEN=*)  . Translation of Unicode string into multibyte character string from the current codepage.

FLAGS  Optional, input. INTEGER (4)  . If specified, argument to modify the string conversion. If FLAGS is omitted, no extra checking of the conversion takes place. Available values are:

   NLS$CompositeCheck  Convert composite characters to precomposed.
   NLS$SepChars  Generate separate characters.
   NLS$DiscardDns  Discard non-spacing characters.
   NLS$DefaultChars  Replace exceptions with default character.

The last three flags (NLS$SepChars, NLS$DiscardDns, and NLS$DefaultChars) are mutually exclusive and can be used only if NLS$CompositeCheck is set, in which case one (and only one) of them is combined with NLS$CompositeCheck using an inclusive OR (IOR or OR). These flags determine what translation to make when there is no precomposed mapping for a base character/nonspace character combination in the Unicode wide character string. The default (IOR (NLS$CompositeCheck, NLS$SepChars)) is to generate separate characters.

Output  
If no error occurs, returns the number of bytes written to MBSTR (bytes are counted, not characters), or the number of bytes required to hold the output string if MBSTR has zero length. If MBSTR is longer than the translation, it is blank-padded. If MBSTR is zero length, the function returns the number of bytes required to hold the translation and nothing is written to MBSTR.
If an error occurs, one of the following negative values is returned:

NLS$ErrorInsufficientBuffer
   The MBSTR argument is too small, but not zero length so that the needed number of bytes is returned.

NLS$ErrorInvalidFlags
   The FLAGS argument has an illegal value.

**MBCS Fortran Equivalent Procedures**

**MBINCHARQQ**

*Same as INCHARQQ except that it can read a single multi-byte character at once and returns the number of bytes read*

**Prototype**

```fortran
INTERFACE
   INTEGER(4) FUNCTION MBInCharQQ(STRING)
   CHARACTER(LEN=2), INTENT(OUT)::STRING
      ! LEN=MBLENMAX
END FUNCTION
END INTERFACE

STRING Output. CHARACTER(MBlenMax). String containing the read characters, padded with blanks up to the length MBlenMax. The MBlenMax parameter, defined in the module iflport.F90, is the longest length, in bytes, of any character in any codepage installed on the system.
Description
Performs the same function as INCHARQQ except that it can read a single multibyte character at once, and it returns the number of bytes read as well as the character.

Output
Number of characters read.

MBINDEX
Same as INDEX, except that multibyte characters can be included in its arguments

Prototype
INTERFACE
  INTEGER(4) FUNCTION MBIndex(STRING, SUBSTRING, &
                              BACK)
  CHARACTER(LEN=*) , INTENT(IN) :: STRING, SUBSTRING
  LOGICAL(4) , INTENT(IN), OPTIONAL :: BACK
END FUNCTION
END INTERFACE

STRING Input. CHARACTER(LEN=*). String to be searched for the presence of SUBSTRING. Can contain multibyte characters.

SUBSTRING Input. CHARACTER(LEN=*). Substring whose position within STRING is to be determined. Can contain multibyte characters.

BACK Optional, input. LOGICAL(4). If specified, determines direction of the search. If BACK is .FALSE. or is omitted, the search starts at the
beginning of STRING and moves toward the end. If BACK is .TRUE., the search starts end of STRING and moves toward the beginning.

Description
Performs the same function as INDEX except that the strings manipulated can contain multibyte characters.

Output
If BACK is omitted or is .FALSE., returns the leftmost position in STRING that contains the start of SUBSTRING. If BACK is .TRUE., returns the rightmost position in STRING which contains the start of SUBSTRING. If STRING does not contain SUBSTRING, returns 0. If SUBSTRING occurs more than once, returns the starting position of the first occurrence (“first” is determined by the presence and value of BACK).

**MBLGE, MBLGT, MBLLE, MBLLT, MBLEQ, MBLNE**

Same as **LGE, LGT, LLE, LLT, **and **the logical operators** .EQ. and .NE..

Prototype

```plaintext
INTERFACE
  LOGICAL(4) FUNCTION MBLGE(STRA, STRB, FLAGS)
  CHARACTER(LEN=*) , INTENT(IN) :: STRA, STRB
  INTEGER(4) , INTENT(IN) , OPTIONAL :: FLAGS
END FUNCTION
END INTERFACE
```

STRA, STRB Input. CHARACTER (LEN=*). Strings to be compared. Can contain multibyte characters.
FLAGS

Optional, input. INTEGER(4). If specified, determines which character traits to use or ignore when comparing strings. You can combine several flags using an inclusive OR (IOR or OR). There are no illegal combinations of flags, and the functions may be used without flags, in which case all flag options are turned off. The available values are:

NLS$MB_IgnoreCase
    Ignore case.

NLS$MB_IgnoreNonspace
    Ignore nonspacing characters (this flag removes Japanese accent characters if they exist).

NLS$MB_IgnoreSymbols
    Ignore symbols.

NLS$MB_IgnoreKanaType
    Do not differentiate between Japanese Hiragana and Katakana characters (corresponding Hiragana and Katakana characters will compare as equal).

NLS$MB_IgnoreWidth
    Do not differentiate between a single-byte character and the same character as a double byte.

NLS$MB_StringSort
    Sort all symbols at the beginning, including the apostrophe and hyphen (See NOTE that follows).
Description
Perform the same functions as LGE, LGT, LLE, LLT and the logical operators .EQ. and .NE. except that the strings being compared can include multi-byte characters, and optional flags can modify the comparison. All these routines have the same arguments as shown for MBLGE above.

NOTE. If the strings supplied contain Arabic Kashidas, the Kashidas are ignored during the comparison. Therefore, if the two strings are identical except for Kashidas within the strings, the functions return a value indicating they are “equal” in the collation sense, though not necessarily identical.

NOTE. When not using the NLS$MB_StringSort flag, the hyphen and apostrophe are special symbols and are treated differently than others. This is to ensure that words like coop and co-op stay together within a list. All symbols, except the hyphen and apostrophe, sort before any other alphanumeric character. If you specify the NLS$MB_StringSort flag, hyphen and apostrophe sort at the beginning also.

Output
Comparisons are made using the current locale, not the current codepage. The codepage used is the default for the language/country combination of the current locale.

The outputs of these functions are as follows:
- MBLGE returns .TRUE. if the strings are equal or STRA comes last in the collating sequence. Otherwise, it returns .FALSE..
- MBLGT returns .TRUE. if STRA comes last in the collating sequence. Otherwise, it returns .FALSE.
• MBLLE returns .TRUE. if the strings are equal or STRA comes first in the collating sequence. Otherwise, it returns .FALSE.
• MBLLT returns .TRUE. if STRA comes first in the collating sequence. Otherwise, it returns .FALSE.
• MBLEQ returns .TRUE. if the strings are equal in the collating sequence. Otherwise, it returns .FALSE.
• MBLNE returns .TRUE. if the strings are not equal in the collating sequence. Otherwise, it returns .FALSE.
• If the two strings are of different lengths, they are compared up to the length of the shortest one. If they are equal to that point, then the return value indicates that the longer string is greater.
• If FLAGS is invalid, the functions return .FALSE.

MBSCAN
Same as SCAN, except that multi-byte characters can be included in its arguments

Prototype
INTERFACE
   INTEGER(4) FUNCTION MBScan(STRING, SET, BACK)
   CHARACTER(LEN=*) , INTENT(IN)::STRING, SET
   LOGICAL(4), INTENT(IN), OPTIONAL::BACK
END FUNCTION
END INTERFACE

STRING    Input. CHARACTER (LEN=*). String to be searched for the presence of any character in SET.
SET       Input. CHARACTER (LEN=*). Characters to search for.
BACK      Optional, input. LOGICAL(4). If specified, determines direction of the search. If BACK is .FALSE. or is omitted, the search starts at the
beginning of STRING and moves toward the end. If BACK is .TRUE., the search starts end of STRING and moves toward the beginning.

Description
Performs the same function as SCAN except that the strings manipulated can contain multibyte characters.

Output
If BACK is .FALSE. or is omitted, returns the position of the leftmost character in STRING that is in SET. If BACK is .TRUE., returns the rightmost character in STRING that is in SET. If no characters in STRING are in SET, returns 0.

MBVERIFY
Same as VERIFY, except that multibyte characters can be included in its arguments

Prototype

```fortran
INTERFACE
  INTEGER(4) FUNCTION MBVerify(STRING, SET, BACK)
  CHARACTER(LEN=*), INTENT(IN)::STRING, SET
  LOGICAL(4), INTENT(IN), OPTIONAL::BACK
END FUNCTION
END INTERFACE

STRING Input. CHARACTER (LEN=*). String to be searched for presence of any character not in SET.
SET Input. CHARACTER (LEN=*). Set of characters tested to verify that it includes all the characters in string.
```
BACK  Optional, input. LOGICAL (4). If specified, determines direction of the search. If BACK is .FALSE. or is omitted, the search starts at the beginning of STRING and moves toward the end. If BACK is .TRUE., the search starts end of STRING and moves toward the beginning.

Description
Performs the same function as VERIFY except that the strings manipulated can contain multibyte characters.

Output
If BACK is .FALSE. or is omitted, returns the position of the leftmost character in STRING that is not in SET. If BACK is .TRUE., returns the rightmost character in STRING that is not in SET. If all the characters in STRING are in SET, returns 0.

**MBJISTToJMS**

*Converts a Japan Industry Standard (JIS) character to a Microsoft Kanji (Shift JIS or JMS) character*

**Prototype**

```fortran
INTERFACE
  CHARACTER (LEN=2) FUNCTION MBJISTToJMS (CHAR)
  CHARACTER (LEN=2), INTENT(IN) :: CHAR
END FUNCTION
END INTERFACE
```

CHAR  Input. CHARACTER (LEN=2). JIS character to be converted.
**Description**

Converts a Japan Industry Standard (JIS) character to a Microsoft Kanji (Shift JIS or JMS) character.

A JIS character is converted only if the lead and trail bytes are in the hexadecimal range 21 through 7E.

---

**NOTE.** *Only computers with Japanese installed as one of the available languages can use the MBJISToJMS conversion function.*

---

**Output**

`MBJISToJMS` returns a Microsoft Kanji (Shift JIS or JMS) character.

---

**MBJMSTToJIS**

*Converts a Microsoft Kanji (Shift JIS or JMS) character to a Japan Industry Standard (JIS) character*

---

**Prototype**

```fortran
INTERFACE
   CHARACTER(LEN=2) FUNCTION MBJMSToJIS(CHAR)
   CHARACTER(LEN=2), INTENT(IN) :: CHAR
END FUNCTION
END INTERFACE
```

**Input.** `CHAR` CHARACTER (LEN=2). JMS character to be converted.

**Description**

Converts a Microsoft Kanji (Shift JIS or JMS) character to a Japan Industry Standard (JIS) character.
A JMS character is converted only if the lead byte is in the hexadecimal range 81 through 9F or E0 through FC, and the trail byte is in the hexadecimal range 40 through 7E or 80 through FC.

**NOTE.** Only computers with Japanese installed as one of the available languages can use the MBJMSToJIS conversion function.

**Output**

MBJMSToJIS returns a Japan Industry Standard (JIS) character.
This chapter describes the functions that comprise the POSIX* library (libPOSF90.lib). These functions are made available to the compiler when you invoke the /4Yposixlib option for Win32* systems, and -posixlib option for Linux systems. These functions implement the IEEE* POSIX FORTRAN-77 Language bindings, as specified in IEEE Standard 1003.9-1992. The POSIX standard is ISO/IEC 9945-1:1990. Copies of the standard are available from IEEE.

The prototypes are described using the INTERFACE call, which provides the required information to complete a call to the specified procedure. For descriptions of the INTERFACE block and the /4Yposixlib option for Win32 systems, and -posixlib option for Linux* systems, see the Intel® Fortran Compiler User's Guide.

**POSIX Library Interface**

Depending on whether you are using free- or fixed-form source, you can interface to POSIX library in the following ways:

- With free-form source, to interface to libPOSF90.lib (POSIX library), your code should contain the

  USE iflposix

  statement where iflposix.f90 is the interface file.
For fixed-form source, the iflposix.f90 file has to be edited and compiled with the /4L132 option for Win32 systems, or -132 for Linux systems. You can include in your fixed-form source the USE iflposix statement, which is not required, but can improve error checking.

**IPXFARGC**

*Returns index of last command-line argument.*

**Prototype**

```
INTERFACE
   INTEGER FUNCTION IPXFARGC()
   END FUNCTION
END INTERFACE
```

**Description**

The function IPXFARGC() returns the number of command-line arguments, excluding the command name, in the command used to invoke the executing program. A return value of zero indicates there are no command-line arguments other than the command name itself.

See also PXFGETARG.
**IPXFWEXITSTATUS**

*Returns the current status of the child process*

**Prototype**

```
INTERFACE
  LOGICAL(4) FUNCTION IPXFWEXITSTATUS (ISTAT)
  INTEGER(4) ISTAT
END FUNCTION IPXFWEXITSTATUS
END INTERFACE
```

**ISTAT**

Output. An INTEGER*4 variable.

**Description**

Returns the exit status of a child process created via PXFFORK. On input, ISTAT contains the process ID of the child process.

**Output**

The function returns .TRUE. if ISTAT is equal to zero, and .FALSE. if ISTAT is not zero.

**IPXFWSTOPSIG**

*Gets the number of the signal that caused the child process to stop*

**Prototype**

```
INTERFACE
  LOGICAL(4) FUNCTION IPXFWSTOPSIG (ISTAT)
  INTEGER(4) ISTAT
END INTERFACE
```
END FUNCTION IPXFWSTOPSIG
END INTERFACE

ISTAT       Output. An integer value representing the child process
            ID on input, and the value of the signal that terminated
            the child process on output.

Description
This function gets the number of the signal that caused the child process to
stop.

Output
The function returns .TRUE. if ISTAT is equal to zero, and .FALSE. if
ISTAT is not zero.

IPXFWTERMSIG

*Gets the number of the signal that
caused termination of a child process*

Prototype
INTERFACE
  LOGICAL(4) FUNCTION IPXFWTERMSIG (ISTAT)
  INTEGER(4) ISTAT
END FUNCTION IPXFWTERMSIG
END INTERFACE

ISTAT       Input/output. Process ID on input, signal number on
            output.

Description
This function gets the number of the signal that caused termination of a
child process defined by the process ID in ISTAT on input.
Output
The function returns .TRUE. if ISTAT is equal to zero, and .FALSE. if ISTAT is not zero.

PXFACCESS
Determines the accessibility of the file

Prototype
INTERFACE
  SUBROUTINE PXFACCESS (PATH, ILEN, IAMODE, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, IAMODE, IERROR
  END SUBROUTINE PXFACCESS
END INTERFACE

PATH   Name of the file.
ILEN   Length of PATH string.
IAMODE One or more of the following:
  for Win32 systems:
    0 -- checks for existence of the file
    2 -- checks for write access
    4 -- checks for read access
    6 -- checks for read/write access
  for Linux systems:
    0 -- checks for existence of the file
    1 -- checks for execute permission
    2 -- checks for write access
    4 -- checks for read access.
    6 -- checks for read/write access
IERROR Return value.
Description
Checks for the accessibility of a file or directory. On Win32 systems, if the name given is a directory name, then the function only checks for existence. All directories have read/write access on Win32 systems.

Output
If successful, IERROR is set to zero. If the file does not exist, or the appropriate access is not available, IERROR is set to −1.

If the file does not exist, or the appropriate access is not available, an error code is returned in IERROR. Possible error codes include:

-1 a bad parameter was passed
EACCES access requested was denied
ENOENT file did not exist

See your Microsoft Visual C++ installation in the include directory under errno.h for the values of EACCES and ENOENT.

PXFAINTGET

*Gets an integer array component of a structure*

Prototype

```fortran
INTERFACE
   SUBROUTINE PXFAINTGET (JHANDLE, COMPNAME, VALUE, IALEN, IERROR)
      INTEGER(4) JHANDLE, IALEN, IERROR
      CHARACTER(LEN=*) COMPNAME
      INTEGER(4) VALUE(IALEN)
   END SUBROUTINE PXFAINTGET
END INTERFACE
```

JHANDLE Handle to the structure.
COMPNAME Component name.
VALUE Output. The value of the component here.
IALEN Length of the VALUE array.
IERROR Return value.

Description
This subroutine gets an integer array component of a structure.

Output
If successful, IERROR is set to zero. ENONAME is set when no such
component name exist for the structure. EARRAYLEN contains the number
of array elements that exceeds IALEN.

PXFAINTSET
Sets an integer array component of a structure

Prototype
INTERFACE
   SUBROUTINE PXFAINSET(JHANDLE, COMPNAME, VALUE,
   IALEN, IERROR)
   INTEGER(4) JHANDLE, IALEN, IERROR
   CHARACTER(LEN=*) COMPNAME
   INTEGER(4) VALUE(IALEN)
   END SUBROUTINE PXFAINTSET
END INTERFACE

JHANDLE Handle to structure.
COMPNAME Component name.
VALUE Values to set.
**IALEN** Length of the VALUE array.

**IERROR** Return value.

**Description**
This subroutine sets an integer array component of a structure.

**Output**
If successful, **IERROR** is set to zero. Otherwise, **ENONAME** is set when no such component name is found in the structure. **EARRAYLEN** is set to the number of array elements that exceeds **IALEN**.

---

**PXF CALL SUBHANDLE**

*Calls the associated subroutine*

**Prototype**

```fortran
INTERFACE
    SUBROUTINE PXFCALLSUBHANDLE(JHANDLE2, IVAL, IERROR)
        INTEGER(4) JHANDLE2, IVAL, IERROR
    END SUBROUTINE PXFCALLSUBHANDLE
END INTERFACE
```

**JHANDLE2** Handle to subroutine.

**IVAL** Argument to subroutine.

**IERROR** Output. Error status.
Description
This subroutine, given a subroutine handle, calls the associated subroutine.

NOTE. The subroutine shall not be a function, an intrinsic, or an entry point and shall be defined with exactly ONE integer argument.

Output
If successful, IERROR is set to zero.

PXFCCHDIR
Changes the current working directory

Prototype
INTERFACE
  SUBROUTINE CHDIR(PATH, ILEN, IERROR)
  CHARACTER(LEN=*) PATH
  INTEGER(4) ILEN, IERROR
END SUBROUTINE CHDIR
END INTERFACE

PATH       The directory to be changed to.
ILEN       Length of the PATH string.
IERROR     Return value.

Description
This subroutine changes the current working directory.

Output
If successful, IERROR is set to zero.
PXFCRCHMOD

Changes the ownership mode of the file.

Prototype

```fortran
INTERFACE
  SUBROUTINE PXFCRCHMOD (PATH, ILEN, IMODE, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, IMODE, IERROR
  END SUBROUTINE PXFCRCHMOD
END INTERFACE
```

<table>
<thead>
<tr>
<th>PATH</th>
<th>Name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILEN</td>
<td>Length of the PATH string.</td>
</tr>
<tr>
<td>IMODE</td>
<td>Mode; on Win32 systems, see your Microsoft Visual C++ Installation in the \include directory under \sys\stat.h for the values of IMODE. On Linux systems, use octal file-access code.</td>
</tr>
<tr>
<td>IERROR</td>
<td>Return value.</td>
</tr>
</tbody>
</table>

Description

This subroutine changes the ownership mode of a file.

**NOTE.** On Linux systems, you must have sufficient ownership permissions, such as being the owner of the file or having read/write access of the file

Output

If successful, IERROR is set to zero.
**PXFCWOWN**

*Changes the owner and group of a file*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFCWOWN(PATH, ILEN, IOWNER, IGROUP, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, IOWNER, IGROUP, IERROR
  END SUBROUTINE PXFCWOWN
END INTERFACE
```

**PATH**  
File or directory name.

**ILEN**  
Length of the **PATH** string.

**IOWNER**  
Owner uid.

**IGROUP**  
Group gid.

**IERROR**  
Output. Error status.

**Description**

This subroutine changes the owner and group of a file.

**Output**

If successful, **IERROR** is set to zero for Linux systems; for Win32 a non-zero value is returned.
**PXFCLOSE**

*Closes the file associated with descriptor*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFCLOSE (FD, IERROR)
    INTEGER(4) FD, IERROR
  END SUBROUTINE PXFCLOSE
END INTERFACE
```

*FD*  
Pile descriptor to be deleted.

*IERROR*  
Output. Returned error code.

**Description**

This subroutine closes the file associated with descriptor.

**Output**

If successful, IERROR is set to zero.

**PXFCLOSEDIR**

*Closes the directory stream*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFCLOSEDIR (IDIRID, IERROR)
    INTEGER(4) IDIRID, IERROR
  END SUBROUTINE PXFCLOSEDIR
END INTERFACE
```

*IDIRID*  
Pointer to DIR structure.
IERROR  Output. Error status.

**Description**
This subroutine closes the directory stream and frees the DIR structure.

**Output**
If successful, IERROR is set to zero.

---

**PXFCONST**

*Retrieves the value associated with a constant*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFCONST (CONSTNAME, IVAL, IERROR)
    CHARACTER(LEN=*)  CONSTNAME
    INTEGER(4)  IVAL, IERROR
  END SUBROUTINE PXFCONST
END INTERFACE
```

**CONSTNAME**  Name of the constant.
**IVAL**  Output. value of the constant.
**IERROR**  Output. return value.

**Description**
This subroutine retrieves the value associated with a constant name. You can inquire about the values of the following symbolic constants:

- STDIN_UNIT
- STDOUT_UNIT
- STDERR_UNIT
- EINVAL
- ENONAME
ENOHANDLE
EARRAYLEN
The constants beginning with \texttt{E} signify various error values for the system variable \texttt{errno}. Check your MSVC++ documentation for more information, on Win32 systems. On Linux, look at the \texttt{/usr/include/errno.h} file.

\textbf{Output}
If successful, \texttt{IERROR} is set to zero; otherwise, it is set to \texttt{-1}.

\section*{PXFCREAT}

\textit{Creates a new file or rewrites an existing one}

\textbf{Prototype}

\begin{verbatim}
INTERFACE
  SUBROUTINE PXFCREAT (PATH, ILEN, IMODE, IFILDES, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, IMODE, IFILDES, IERROR
  END SUBROUTINE PXFCREAT
END INTERFACE
\end{verbatim}

\textbf{PATH} \hspace{1cm} Pathname of the file.
\textbf{ILEN} \hspace{1cm} Length of \texttt{PATH} string.
\textbf{IMODE} \hspace{1cm} Mode of the newly created file; on Win32 systems see your Microsoft Visual C++ Installation in the \texttt{\include} directory under \texttt{sys\stat.h} for the values of \texttt{IMODE}. On Linux systems, use octal file-access code.
\textbf{IFILDES} \hspace{1cm} Output. file descriptor.
\textbf{IERROR} \hspace{1cm} Return value.
Synopsis

This subroutine creates a new file or rewrites an existing one.

Output

If successful, IERROR is set to zero.

PXFDUP

Duplicates an existing file descriptor

Prototype

INTERFACE
  SUBROUTINE PXFDUP (IFILDES, IFID, IERROR)
  INTEGER(4) IFILDES, IFID, IERROR
END SUBROUTINE PXFDUP
END INTERFACE

IFILDES  Descriptor to be duplicated.
IFID     Output. handle for the duplicated descriptor.
IERROR   Output. returned error code.

Description

This subroutine duplicates an existing file descriptor.

Output

If successful, IERROR is set to zero.
PXFDUP2

Duplicates an existing file descriptor

Prototype

INTERFACE
SUBROUTINE PXFDUP2 (IFILDES, IFILDES2, IERROR)
INTEGER(4) IFILDES, IFILDES2, IERROR
END SUBROUTINE PXFDUP2
END INTERFACE

IFILDES    Descriptor to be duplicated.
IFILDES2   Desired handle for the duplicated descriptor.
IERROR     Output. Returned error code.

Description
This subroutine duplicates an existing file descriptor.

Output
If successful, IERROR is set to zero.

PXFEINTGET

Gets a single element from an integer array component of a structure

Prototype

INTERFACE
SUBROUTINE PXFEINTGET (JHANDLE, COMPNAME, INDEX, VALUE, IERROR)
DESCRIPTION
This subroutine gets a single element from an integer array component of a structure.

Output
If successful, IERROR is set to zero. Otherwise, ENONAME is set when EINVAL is set if an invalid index is specified.

PXFEINTGET

Sets a single element from an integer array component of a structure

Prototype

INTERFACE
  SUBROUTINE PXFEINTGET (JHANDLE, COMPNAME, IVALUE, INDEX, IERROR)
  CHARACTER(LEN=*) COMPNAME
  INTEGER(4) JHANDLE, IVALUE, INDEX, IERROR
END SUBROUTINE PXFEINTGET

END INTERFACE

JHANDLE Handle to structure.
COMPNAME Component name.
INDEX Index of element.
VALUE Output. Address where the value of the component is stored.
IERROR Return value.
END INTERFACE

JHANDLE Handle to structure.
COMPNAME Component name.
IVALUE Index of element.
INDEX Value to set.
IERROR Return value.

Description
This subroutine sets a single element from an integer array component of a structure.

Output
If successful, IERROR is set to zero. Otherwise, ENONAME is set when no such component name exist for the structure. EINVAL is set when an invalid index is specified.

PXFESTRGET
*Gets a single element from a string array component of a structure*

Prototype

INTERFACE
SUBROUTINE PXFESTRGET (JHANDLE, COMPNAME, INDEX, VALUE, ILEN, IERROR)
INTEGER(4) JHANDLE, INDEX, ILEN, IERROR
CHARACTER(LEN=*) COMPNAME, VALUE
END SUBROUTINE PXFESTRGET
END INTERFACE

JHANDLE Handle to structure.
COMPNAME  Component name.
INDEX     Index of element.
VALUE     Output where the string is stored.
ILEN      Output. Length of string.
IERROR    Return value.

Description
This subroutine gets a single element from a string array component of a
structure.

Output
If successful, IERROR is set to zero. Otherwise, ENONAME is set when no
such component name exist for the structure. EINVAL is set when an
invalid index is specified.

PXFEXECV
Executes an executable

Prototype
INTERFACE
  SUBROUTINE PXFEXECV (PATH, LENPATH, ARGV,
                        LENARGV, IARGC, IERROR)
  INTEGER(4) LENPATH, LENARGV(0:IARGC-1), IARGC,
            IERROR
  CHARACTER(LEN=*) PATH, ARGV(0:IARGC-1)
END SUBROUTINE PXFEXECV
END INTERFACE

PATH       Input. New executable path.
LENPATH    Input. Length of PATH string.
LENARGV Input. Length of each argument string.
IARGC Input. Argument count.
IERROR Output. Error value.

Description
This subroutine executes an executable file. If successful, these functions do not return to the calling process.

Output
Zero if successful, otherwise, an error code, the value of ierrno.

PXFEXECVE

*Executes an executable with environment settings*

Prototype

```
INTERFACE
  SUBROUTINE PXFEXECVE (PATH, LENPATH, ARGV, LENARGV, IARGC, ENV, LENENV, IENVC, IERROR)
  INTEGER(4) LENPATH, LENARGV(0:IARGC-1), IARGC,
           LENENV(IENVC), IENVC, IERROR
  CHARACTER(LEN=*) PATH, ARGV(0:IARGC-1), ENV(IENVC)
  END SUBROUTINE PXFEXECVE
END INTERFACE
```

PATH Input. New executable path.
LENPATH Input. Length of PATH string.
LENARGV Input. Length of each argument string.
IARGC    Input. Argument count.
ENV      Input. Environment setting.
LENENV   Input. Length of elements in ENV.
IENVC    Input. Number of elements in ENV.
IERROR   Return value.

Description
This subroutine executes an executable file with environment settings.

Output
None.

PXFEEXECVP
Executes a file

Prototype
INTERFACE
SUBROUTINE PXFEEXECVP (FILE, LENFILE, ARGV, LENARGV, IARGC, IERROR)
INTEGER(4) LENFILE, LENARGV(0:IARGC-1), IARGC, IERROR
CHARACTER(LEN=*) FILE, ARGV(0:IARGC-1)
END SUBROUTINE PXFEEXECVP
END INTERFACE

FILE    Input. File to be executed.
LENFILE Input. Length of FILE.
LENARGV Input. Length of each argument string.
IARGC   Input. Argument count.
**IERROR**

Output. Error value.

**Description**

Executes a file.

**Output**

None.

---

**PXFEXIT**

*Exits from a process*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFEXIT (ISTATUS)
    INTEGER(4) ISTATUS
  END SUBROUTINE PXFEXIT
END INTERFACE
```

**ISTATUS**

Input. Exit value.

**Description**

This subroutine exits a process.

**Output**

None.
PXFFASTEXIT

Exits from the process

Prototype

INTERFACE
  SUBROUTINE PXFFASTEXIT (ISTATUS)
  INTEGER(4) ISTATUS
  END SUBROUTINE PXFFASTEXIT
END INTERFACE
ISTATUS Input. Exit value.

Description
This subroutine exits from the process. There is no possible return from this subroutine.

Output
None.

PXFFFLUSH

Writes any unwritten data for an output stream

Prototype

INTERFACE
  LOGICAL(4) FUNCTION PXFFFLUSH (LUNIT, IERROR)
  INTEGER(4) IUNIT, IERROR
  END FUNCTION PXFFFLUSH
END INTERFACE
LUNIT Input. A FORTRAN logical unit, which must be an INTEGER*4 expression with the value in the range of 0 to 100.

IERROR Output. Return value.

**Description**

This subroutine writes any unwritten data for an output stream.

**Output**

If successful, IERROR is set to zero. For Linux, IERROR is always set to -1 for compatibility only as this function is not used for Linux.

---

**PXFFGETC**

*Reads a character from the specified stream (logical unit)*

**Prototype**

```
INTERFACE
  SUBROUTINE PXFFGETC (LUNIT, CHAR, IERROR)
  CHARACTER(L=1) CHAR
  INTEGER(4) IERROR
  END SUBROUTINE PXFFGETC
END INTERFACE
```

LUNIT A FORTRAN logical unit, which must be an INTEGER*4 expression with the value in the range of 0 to 100.

CHAR Character to be read.

IERROR Return value.
Description
This subroutine reads a character from the specified stream (logical unit).

Output
If successful, IERROR is set to zero.

PXFFILENO

Returns the file descriptor associated with a stream

Prototype

INTERFACE
   SUBROUTINE PXFFILENO (LUNIT, FD, IERROR)
   INTEGER(4) LUNIT, FD, IERROR
   END SUBROUTINE PXFFILENO
END INTERFACE

LUNIT  a FORTRAN logical unit, which must be an INTEGER*4 expression with the value in the range of 0 to 100.
FD  Output. File descriptor.
IERROR  Output. Returned error code.

Description
This subroutine returns the file descriptor associated with a stream.

Output
If successful, IERROR is set to zero. Otherwise, EINVAL is set if LUNIT is not an open unit. EBADF is set if LUNIT is not connected with a file descriptor.
PXFFORK

Fork a child process

Prototype for Win32

INTERFACE

SUBROUTINE PXFFORK (IPID, PATHNAME, ARGS, IERROR)
  INTEGER(4) IPID, IERROR
  CHARACTER(LEN=*) , INTENT(IN) :: PATHNAME
  CHARACTER(LEN=*) ARGS(:)
END SUBROUTINE PXFFORK
END INTERFACE

Prototype for Linux

INTERFACE PXFFORK

SUBROUTINE LINUX_PXFFORK (IPID, IERROR)
  INTEGER(4) IPID, IERROR
END SUBROUTINE LINUX_PXFFORK

SUBROUTINE PXFFORK(IPID, PATHNAME, ARGS, IERROR)
  INTEGER(KIND=4), INTENT(OUT):: IPID
  CHARACTER(LEN=*) , INTENT(IN) :: PATHNAME
  CHARACTER(LEN=*) ARGS(:)
  INTEGER(KIND=4),INTENT(OUT):: IERROR
END SUBROUTINE
END INTERFACE

IPID Output. Process ID.
PATHNAME Input. Path to executable file. Variable is used only for Win32 systems.
ARGS Input. Arguments for executable file. Variable is used only for Win32 systems.
POSIX Functions

IERROR

Output. Zero if the process is started successfully. Otherwise contains an appropriate error value of IERRNO.

Description
This subroutine spawns an asynchronous child process. Interface that includes arguments PATHNAME and ARGS is used only in Win32 systems.

Output
A process handle or process ID is returned in the variable IPID if the process was successfully forked. Otherwise, the returned IPID value is -1.

PXFFPUTC

Outputs a character to the specified stream (logical unit)

Prototype

INTERFACE
  SUBROUTINE PXFFPUTC (LUNIT, CHAR, IERROR)
  CHARACTER(LEN=1) CHAR
  INTEGER(4) LUNIT, IERROR
  END SUBROUTINE PXFFPUTC
END INTERFACE

LUNIT

A FORTRAN logical unit, which must be an INTEGER*4 expression with the value in the range of 0 to 100.

CHAR

Character to be written.

IERROR

Return value.

Description
This subroutine outputs a character to the specified stream.
Output

If successful, IERROR is set to zero. EEND is set if end of file is reached.

**PXFFSEEK**

*Sets the position of the next input / output on the stream*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFFSEEK (LUNIT, IOFFSET, IWHENCE, IERROR)
    INTEGER(4) LUNIT, IOFFSET, IWHENCE, IERROR
END SUBROUTINE PXFFSEEK
END INTERFACE
```

- **LUNIT**: A Fortran logical unit, which must be an INTEGER*4 expression with the value in the range of 0 to 100.
- **IOFFSET**: Number of bytes away from whence to place the pointer.
- **IWHENCE**: Determines the position within the stream as one of the following constants, which are defined in STDIO.H for Win32 or Linux systems:
  - SEEK_SET =0 offset from beginning of file.
  - SEEK_CUR =1 offset from current position of file pointer.
  - SEEK_END =2 offset from end of file.
- **IERROR**: Return value.

**Description**

This subroutine sets the position of the next input/output on the stream.
Output

If successful, IERROR is set to 0. EINVAL is set if no file connected to LUNIT if IWHENCE is not a proper value or resulting offset would be invalid. ESPipe is set if LUNIT is a pipe or FIFO. EEND is set if end of file is reached.

PXFFSTAT

Gets a file status

Prototype

INTERFACE
  SUBROUTINE PXFFSTAT (IFILDES, JSTAT, IERROR)
  INTEGER(4) IFILDES, JSTAT, IERROR
  END SUBROUTINE PXFFSTAT
END INTERFACE

IFILDES File handle.
JSTAT Pointer to the PXFFSTAT structure.
IERROR Return value.

Description

This subroutine gets the status of a file.

Output

If successful, IERROR is set to zero.
PXFFTELL

*Returns the relative position in bytes from the beginning of the file*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFFTELL (LUNIT, IOFFSET, IERROR)
  INTEGER(4) LUNIT, IOFFSET, IERROR
  END SUBROUTINE PXFFTELL
END INTERFACE
```

LUNIT A Fortran logical unit, which must be an INTEGER*4 expression with the value in the range of 0 to 100.

IOFFSET Output. Relative position in bytes from beginning of the file.

IERROR Return value.

**Description**

This subroutine is used for Win32 only. It returns the relative position in bytes from the beginning of the file.

**Output**

If successful, IERROR is set to zero.
PXGETARG
Get the specified command-line argument

Prototype
INTERFACE
  SUBROUTINE (ARGNUM, STR, ISTR, IERROR)
    CHARACTER (LEN=*) STR
    INTEGER (4) ARGNUM, ISTR, IERROR
  END SUBROUTINE PXGETARG
END INTERFACE
ARGNUM Input. Number of command-line argument.
STR Input/Output. Place for required argument.
ISTR Input. Length of place for required argument.
IERROR Output. Error value.

Description
This subroutine returns command-line argument. See also the IPXFARGC function.

Output
If successful, IERROR is set to zero, STR is set to command-line argument.
LEN is set to length of returned command-line argument.
PXFGETC
Reads a character from standard input, unit 5

Prototype

INTERFACE
  SUBROUTINE PXFGETC (NEXTCHAR, IERROR)
    CHARACTER(LEN=1) NEXTCHAR
    INTEGER(4) IERROR
  END SUBROUTINE PXFGETC
END INTERFACE

NEXTCHAR Character to be read.
IERROR Return value.

Description
This subroutine reads a character from the standard input, unit 5.

Output
If successful, IERROR is set to zero.

PXFGETCWD
Retrieves the path of the current working directory

Prototype

INTERFACE
  SUBROUTINE PXFGETCWD (BUF, ILEN, IERROR)
POSIX Functions

CHARACTER(LEN=*) BUF
INTEGER(4) ILEN, IERROR
END SUBROUTINE PXFGETCWD
END INTERFACE

BUF         Output. Storage for the retrieved pathname.
ILEN        Output. Length of the retrieved pathname.
IERROR      return value.

Description
This subroutine retrieves the path of the current working directory.

Output
If successful, IERROR is set to zero. EINVAL is set if the size of BUF is insufficient.

PXFGETGRGID

Gets entry with specified GID

Prototype

INTERFACE
  SUBROUTINE PXFGETGRGID (JGID, JGROUP, IERROR)
  INTEGER(4) IGID, JGROUP, IERROR
  END SUBROUTINE PXFGETGRGID
END INTERFACE

JGID         Group ID.
JGROUP       Output. Pointer to the structure PXFGROUP.
IERROR       Output. Error status.

Description
This subroutine gets entry with specified group ID.
Output
If successful, IERROR is not changed.

**PXGETGRNAM**

*Get entry with the specified group name*

**Prototype**

```
INTERFACE
    SUBROUTINE PXFGETGRNAM (NAME, ILEN, JGROUP, IERROR)
        CHARACTER(LEN=*) NAME
        INTEGER(4) ILEN, JGROUP, IERROR
    END SUBROUTINE PXFGETGRNAM
END INTERFACE
```

**NAME**  
Name of group.

**ILEN**  
Length of NAME string.

**JGROUP**  
Output. Pointer to the structure PXFGROUP.

**IERROR**  
Output. Error status.

**Description**

This subroutine gets entry with the specified group name.

**Output**

If successful, IERROR is not changed.

Under Windows* system, the function is included for compatibility only. It only copies the string NAME to the corresponding field in STRUCTURE pointed by JGROUP.
PXGETPWNAM

Gets entry with specified user name

Prototype

```plaintext
INTERFACE
  SUBROUTINE PXGETPWNAM (NAME, ILEN, JPASSWD, IERROR)
    CHARACTER(LEN=*) NAME
    INTEGER(4) JPASSWD, ILEN, IERROR
  END SUBROUTINE
END INTERFACE
```

NAME
Login name of the user.

ILEN
Length of the NAME string.

JPASSWD
Output. Pointer to the structure PXFPASSWD.

IERROR
Output. Error status.

Description

This subroutine gets entry with specified user name. For example, a login name might be “jsmith” while the actual name is “John Smith.”

Under Windows, this function just copies the string passed by parameter NAME to corresponding field of structure with handle JPASSWD.

Output

If successful, IERROR is not changed.

Under Windows* system, the function is included for compatibility only. It only copies the string NAME to the corresponding field in STRUCTURE pointed by JGROUP.
PXFGETPWUID

*Gets entry with specified UID*

**Prototype**

```fortran
INTERFACE
    SUBROUTINE PXFGETPWUID (IUID, JPASSWD, IERROR)
    INTEGER(4) IUID, JPASSWD, IERROR
END SUBROUTINE PXFGETPWUID
END INTERFACE
```

- **IUID** User ID.
- **JPASSWD** Output. pointer to the structure PXFPASSWD.
- **IERROR** Output. error status.

**Description**

This subroutine gets entry with specified user ID.

Under Windows, this function just copies the string to corresponding field of structure with handle JPASSWD.

**Output**

If successful, IERROR is set to zero.

Under Windows* system, the function is included for compatibility only. It only copies the string “*******” to the corresponding field in STRUCTURE pointed by JPASSWD.
PXFGETSUBHANDLE

*Returns a subroutine handle for a specified subroutine*

**Prototype**

```
INTERFACE
   SUBROUTINE PXFGETSUBHANDLE (SUB, JHANDLE1, IERROR)
      INTEGER(4) JHANDLE1, IERROR
   EXTERNAL SUB
   END SUBROUTINE PXFGETSUBHANDLE
END INTERFACE
```

**Description**

This subroutine returns a subroutine handle for a specified subroutine.

**NOTE.** The argument `SUB` shall not be a function, an intrinsic, or an entry point, and shall be defined with exactly one integer argument.

**Output**

If successful, `IERROR` is set to zero.
PXFINTGET

Gets an integer component of a structure

Prototype

INTERFACE
    SUBROUTINE PXFINTGET (JHANDLE, COMPNAME, VALUE, IERROR)
    INTEGER(4) JHANDLE, VALUE, IERROR
    CHARACTER(LEN=*) COMPNAME
    END SUBROUTINE PXFINTGET
END INTERFACE

JHANDLE Handle to the structure.
COMPNAME Component name.
VALUE Output. where the value of the component is stored.
IERROR Return value.

Description

This subroutine gets an integer component of a structure.

Output

If successful, IERROR is set to zero. ENONAME is set if no such component name for the structure.
**PXFINTSET**

*Sets an integer component of a structure*

**Prototype**

```fortran
INTERFACE
    SUBROUTINE PXFINTSET (JHANDLE, COMPNAME, VALUE, IERROR)
    INTEGER(4) JHANDLE, IERROR, VALUE
    CHARACTER(LEN=*) COMPNAME
    END SUBROUTINE
END INTERFACE
```

**JHANDLE**
Handle to the structure.

**COMPNAME**
Component name.

**VALUE**
Set to this value.

**IERROR**
Return value.

**Description**
This subroutine sets an integer component of a structure.

**Output**
If successful, IERROR is set to zero. ENONAME is set if no such component name for the structure.
PXFISBLK

Tests for block special file

Prototype

INTERFACE
  LOGICAL(4) FUNCTION PXFISBLK (M)
  INTEGER(4) M
END FUNCTION PXFISBLK
END INTERFACE

M          Value of the ST_Mode component in the structure PXFSTAT (stat).

Description

This function tests for block special file. Under Windows, this function just returns -1.

Output

A logical value is returned showing whether the specified file handle corresponds to a block device: .TRUE. if the file is a block mode file, .FALSE. otherwise.

For Windows systems, this function always returns -1 (function not used).
**PXFISCHR**

*Checks if character file*

**Prototype**

```fortran
INTERFACE
   LOGICAL(4) FUNCTION PXFISCHR (M)
   INTEGER(4) m
   END FUNCTION PXFISCHR
END INTERFACE
```

M 
Value of the ST_MODE component in the structure PXFSTAT (stat).

**Description**

This function checks if the file is character file.

**Output**

A logical value is returned, showing whether the specified file handle corresponds to a character mode device: .TRUE. if the logical unit M is connected to a character mode device, .FALSE. otherwise.

**PXFISCONST**

*Retrieves value associated with a constant name*

**Prototype**

```fortran
INTERFACE
   LOGICAL(4) FUNCTION PXFISCONST (M)
   CHARACTER(LEN=*) M
END INTERFACE
```
END FUNCTION PXFISCONST
END INTERFACE

M Input. Name of the constant.

**Description**
Retrieves value associated with a constant name. You can inquire about the values of the following symbolic constants:

- STDIN_UNIT
- STDOUT_UNIT
- STDERR_UNIT
- EINVAL
- ENONAME
- ENOHANDLE
- EARRAYLEN
- ETRUNC

**Output**
A logical value is returned. Returns `.TRUE.` if `M` is one of these constants, `.FALSE.` if not.

---

**PXFISDIR**

*Checks if component is a directory*

**Prototype**

```fortran
INTERFACE
   LOGICAL(4) FUNCTION PXFISDIR (M)
   INTEGER(4) M
   END FUNCTION PXFISDIR
END INTERFACE
```
M Value of the ST_MODE component in the structure PXFSTAT (stat).

Description
This subroutine checks if the component is a directory.

Output
Returns .TRUE. if M is a file handle representing a directory, .FALSE. otherwise.

PXFISFIFO
Checks to determine whether a file is a special FIFO file

Prototype
INTERFACE
  LOGICAL(4) FUNCTION PXFISFIFO (M)
  INTEGER(4) M
END FUNCTION PXFISFIFO
END INTERFACE

M Value of the ST_MODE component in the structure PXFSTAT (stat).

Description
This function checks to determine whether a file is a special FIFO file created by PXFMKFIFO.

NOTE. On Win32 systems, this function always returns an error code of -1. Win32 does not support symbolic file links.
Output
A logical .TRUE./.FALSE. value that indicates whether the file is a special FIFO file.

**PXFISREG**

*Checks if a file is a special FIFO file.*

**Prototype**

```fortran
INTERFACE
   LOGICAL(4) FUNCTION PXFISREG (M)
   INTEGER(4) M
END FUNCTION PXFISREG
END INTERFACE
```

M Value of the ST_MODE component in the structure PXFSTAT (stat).

**Description**

This function checks if a LOGICAL(4) value is .TRUE./.FALSE. to determine if the queried file is a special FIFO file. If PXFISREG returns true, the file is a regular file.

**NOTE.** On Win32 systems, this function always returns an error code of -1. Win32 does not support symbolic file links.

**Output**

A logical .TRUE./.FALSE. value
**PXFKILL**

_Sends a signal to a specified process._

**Prototype**

INTERFACE
  SUBROUTINE PXFKILL (IPID, ISIG, IERROR)
  INTEGER(4) IPID, ISIG, IERROR
  END SUBROUTINE PXFKILL
END INTERFACE

**IPID**

Specifies what process to kill as determined by its value:
- > 0           the specific process is killed.
- < 0           all processes in the group are killed.
- == 0          all processes in the group except special processes are killed.
- == pid_t-1    all processes are killed.

**ISIG**

Value of desired signal.

**IERROR**

Return value.

**Description**

This subroutine sends a signal with value ISIG to a specified process. For Windows systems only, ISIG=9 can be used to kill the specified process.

**Output**

IERROR contains a zero if no error, otherwise nonzero. On Linux systems, if successful, IERROR returns zero, otherwise IERROR = -1.
PXFLINK
Links to a file/directory

Prototype

INTERFACE
  SUBROUTINE PXFLINK (EXISTING, LENEXIST, NEW, &
                     LENNEW, IERROR)
  CHARACTER(LEN=*) EXISTING, NEW
  INTEGER(4) LENEXIST, LENNEW, IERROR
END SUBROUTINE PXFLINK
END INTERFACE

EXISTING       Existing file/directory name.
LENEXIST       Length of EXISTING string.
NEW            New file/directory name.
LENNEW         Length of NEW string.
IERROR          Error status.

Description
Under Windows, this function copies the file with name EXISTING to the
file with name NEW. Windows does not support such feature as a link.
This subroutine links to a file/directory.

Output
If successful, IERROR is set to zero.

Windows does not support such feature as link. Under Windows, this
function only copies the file with the name EXISTING to the file with
name NEW.
PXFLocaltime

Converts a given elapsed time in seconds into current system date

Prototype

INTERFACE
SUBROUTINE PXFLocaltime (STIME, DATEARRAY, IERROR)
INTEGER(4) ISECNDS, DATEARRAY(9), IERROR
END SUBROUTINE PXFLocaltime
END INTERFACE

STIME Elapsed time in seconds since 00:00:00 Greenwich Mean Time, January 1, 1970.

DATEARRAY Will contain current date and system time:
(1) seconds (0-59)
(2) minutes (0-59)
(3) hours (0-23)
(4) day of month (1-31)
(5) month (1-12)
(6) year (Gregorian)(ex. 1990)
(7) day of week (0-6, 0 is Sunday)
(8) day of year (1-366)
(9) daylight savings (1, if in effect; 0, otherwise)

IERROR Return value.

Description
Converts a given elapsed time in seconds into current system date.

Output
If successful, IERROR is set to zero.
**PXFLSEEK**

*This function positions a file a specified distance in bytes*

**Prototype**

```fortran
INTERFACE
    SUBROUTINE PXFLSEEK (IFILDES, OFFSET, WHENCE, POSITION, IERROR)
        INTEGER(4) IFILDES, OFFSET, WHENCE, POSITION, IERROR
    END SUBROUTINE PXFLSEEK
END INTERFACE
```

- **IFILDES**: File descriptor.
- **OFFSET**: Number of bytes to move.
- **WHENCE**: Starting position.
- **POSITION**: Output. ending position.
- **IERROR**: Returned error code.

**Description**

This subroutine positions a file a specified distance (OFFSET) in bytes depending upon the input value of **WHENCE**. Where **WHENCE** is one of the following:

- **SEEK_SET** 0
- **SEEK_CUR** 1
- **SEEK_END** 2

If **WHENCE** is **SEEK_SET**, the file is positioned to **OFFSET** bytes from the beginning of the file. If **WHENCE** is **SEEK_CUR**, the file is positioned to **OFFSET** bytes from the current position of the file. If **WHENCE** is **SEEK_END** the file is positioned **OFFSET** bytes beyond the end of the line.
Output

If successful, IERROR is set to zero.

PXFMKDIR

*Makes a directory*

Prototype

INTERFACE

SUBROUTINE PXFMKDIR (PATH, ILEN, IMODE, IERROR)
CHARACTER(LEN=*) PATH
INTEGER(4) ILEN, IMODE, IERROR
END SUBROUTINE PXFMKDIR
END INTERFACE

PATH The directory to be changed to.
ILEN Length of PATH string.
IMODE Mode mask.
IERROR Return value.

Description

This subroutine creates a directory. IMODE value is processed in the Linux systems only, octal of code of file access is set.

Output

If successful, IERROR is set to zero.
**PXFMKFIFO**

*Creates a new FIFO*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFMKFIFO (PATH, ILEN, IMODE, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, IMODE, IERROR
  END SUBROUTINE
END INTERFACE
```

**PATH**
Name of FIFO file to create.

**ILEN**
Length of PATH string.

**IMODE**
File permission (bits).

**IERROR**
Error status.

**Description**

This subroutine creates a new FIFO.

Under Windows, this function performs just opening of the file. Windows does not support such feature as FIFO.

**Output**

On Linux, if successful, IERROR is set to zero. On Win32 systems if successful, IERROR is not changed.

Windows does not support such feature as FIFO. Under Windows, this function only opens file PATH.
PXFOPEN

Opens/creates a file for reading or writing

Prototype

INTERFACE
  SUBROUTINE PXFOPEN (PATH, ILEN, IOPENFLAG,&
                     IMODE, IFILDES, IERROR)
  CHARACTER(LEN=*) PATH
  INTEGER(4) ILEN, OPENFLAG, IMODE, IFILDES, IERROR
  END SUBROUTINE
END INTERFACE

PATH        Pathname of the file.
ILEN        Length of PATH string.
IOPENFLAG   Specifies how to open the file (read/write).
IMODE       Mode of the newly-created file, if
             IOPENFLAG = O_CREAT
IFILDES     Output. File descriptor.
IERROR      Return value.

Description

This subroutine creates a file or opens an existing one for read/write.

Output

If successful, IERROR is set to zero.
PXFOPENDIR

Opens a directory and associates a stream with it

Prototype

INTERFACE
  SUBROUTINE PXFOPENDIR (DIRNAME, LENDIRNAME, OPENDIRID, IERROR)
    CHARACTER(LEN=*) DIRNAME
    INTEGER(4) LENDIRNAME, OPENDIRID, IERROR
  END SUBROUTINE PXFOPENDIR
END INTERFACE

DIRNAME          Directory name.
LENDIRNAME       Length of DIRNAME string.
OPENDIRID        Output. Pointer to DIR structure.
IERROR           Output. Error status.

Description

This subroutine opens a directory and associates a stream with it.

Output

If successful, IERROR is set to zero.
**PXFPIPE**

*Creates a communications pipe between two processes*

---

**Prototype**

```plaintext
INTERFACE
    SUBROUTINE PXFPIPE (IREADFD, IWRITEFD, IERROR)
    INTEGER(4) IREADFD, IWRITEFD, IERROR
END SUBROUTINE PXFPIPE
END INTERFACE
```

- **IREADFD**
  - Output. Reading file descriptor.
- **IWRITEFD**
  - Output. Writing file descriptor.
- **IERROR**
  - Output. Returned error code.

**Description**

This subroutine creates a communications pipe between two processes.

**Output**

If successful, IERROR is set to zero.

---

**PXFPUTC**

*Outputs a character to logical unit 6 (stdout)*

---

**Prototype**

```plaintext
INTERFACE
    SUBROUTINE PXFPUTC (CH, IERROR)
```

---

3-53
CHARACTER(LEN=1) CH
INTEGER(4) IERROR
END SUBROUTINE PXFPUTC
END INTERFACE

CH Character to be written.
IERROR Returned error code.

Description
This subroutine outputs a character to logical unit 6 (stdout).

Output
If successful, IERROR is set to zero. EEND is set if end of file is encountered.

PXFREAD
Reads from a file

Prototype
INTERFACE
  SUBROUTINE PXFREAD (IFILDES, CHAR, NBYTE, NREAD, IERROR)
  INTEGER(4) IFILDES
  CHARACTER(LEN=*) BUF
  INTEGER(4) NBYTE, NREAD, IERROR
END SUBROUTINE
END INTERFACE

IFILDES Descriptor to be read from.
CHAR Buffer to be written to.
NBYTE Number of bytes to read.
NREAD Output. Number of bytes read.
**IERROR**  Output. Returned error code.

**Description**
This subroutine reads a specified number of bytes from a file.

**Output**
If successful, **IERROR** is set to zero. Otherwise, **NREAD** is undefined if error occurs.

---

**PXFREADDR**
*Reads the current directory entry*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFREADDR (IDIRID, JDIRENT, IERROR)
  INTEGER(4) IDIRID, JDIRENT, IERROR
  END SUBROUTINE PXFREADDR
END INTERFACE
```

**IDIRID**  Pointer to **DIR** structure.

**JDIRENT**  Output. Pointer to **PXFDIRENT** structure.

**IERROR**  Output. Error status.

**Description**
This subroutine reads the current directory entry.

**Output**
If successful, **IERROR** is set to zero.
PXFRENAME

Changes the name of a file

Prototype

INTERFACE
SUBROUTINE PXFRENAME (OLD, LENOLD, NEW, LENNEW, IERROR)
CHARACTER(LEN=*) OLD, NEW
INTEGER(4) LENOLD, LENNEW, IERROR
END SUBROUTINE PXFRENAME
END INTERFACE

OLD          Old filename.
LENOLD       Length of OLD string.
NEW          New filename.
LENNEW       Length of NEW string.
IERROR       Return value.

Description

This subroutine changes the name of file.

Output

If successful, IERROR is set to zero.
**PXFREWINDDIR**

*Resets the position of the stream to the beginning of the directory*

**Prototype**

```
INTERFACE
  SUBROUTINE PXFREWINDDIR (IDIRID, IERROR)
    INTEGER(4) IDIRID, IERROR
  END SUBROUTINE PXFREWINDDIR
END INTERFACE
```

**IDIRID**  
Pointer to DIR structure.

**IERROR**  
Output. Error status.

**Description**

This subroutine resets the position of the stream to the beginning of the directory.

**Output**

If successful, IERROR is set to zero.

---

**PXFRMDIR**

*Removes a directory*

**Prototype**

```
INTERFACE
  SUBROUTINE PXFRMDIR (PATH, ILEN, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, IERROR
  END SUBROUTINE PXFRMDIR
END INTERFACE
```
END SUBROUTINE PXFRMDIR
END INTERFACE

PATH The directory to be removed.
ILEN Length of PATH string.
IERROR Return value.

Description
This subroutine removes a directory.

Output
If successful, IERROR is set to zero.

PXFSIGADDSET
Adds a signal to the set

Prototype
INTERFACE
SUBROUTINE PXFSIGADDSET (JSIGNET, ISIGNO, IERROR)
INTEGER(4) JSIGNSET, ISIGNO, IERROR
END SUBROUTINE PXFSIGADDSET
END INTERFACE

JSIGNET A bit vector structure created via PXFSTRUCTURECREATE.
ISIGNO Input. Signal to be added to the set.
IERROR Return code.
**Description**

Adds a signal to the set JSIGNET.

---

**NOTE.** You must use the PXFSTRUCTURECREATE subroutine to create the JSIGNSET structure. JSIGNSET is not inherited by child applications started as a result of PXFFORK.

---

**Output**

If successful, IERROR is set to zero.

---

**PXFSIGDELSET**

*Deletes a signal from the set*

---

**Prototype**

```plaintext
INTERFACE
    SUBROUTINE PXFSIGDELSET (JSIGSET, ISIGNO, IERROR)
    INTEGER(4) JSIGSET, ISIGNO, IERROR
END SUBROUTINE PXFSIGDELSET
END INTERFACE
```

- **JSIGNET**  
  A set representing a group of signals (obtained with PXFSTRUCTURECREATE).

- **ISIGNO**  
  Input. signal to be deleted to the set.

- **IERROR**  
  Return code.
Description
Deletes the value of a given signal, \texttt{ISIGNO}, from the set \texttt{JSIGNSET}. After the signal has been deleted, any occurrence of the signal will not be “caught” by the application, and will be re-sighaled, causing possible fatal application errors.

Output
If successful, \texttt{IERROR} is set to zero.

\section*{PXFSIGEMPTYSET}

\textit{Determines whether a signal set is empty.}

Prototype

\begin{verbatim}
INTERFACE
   SUBROUTINE PXFSIGEMPTYSET (JSIGSET, IERROR)
      INTEGER(4) JSIGSET, IERROR
   END SUBROUTINE
END INTERFACE
\end{verbatim}

\texttt{JSIGNET} \hspace{1cm} A set representing a group of signals (obtained with \texttt{PXFSSTRUCTURECREATE}).

\texttt{IERROR} \hspace{1cm} Return code.

Description
Tests to determine whether the set given by \texttt{JSIGNSET} is empty.

\textbf{NOTE.} You must use the \texttt{PXFSSTRUCTURECREATE} subroutine to create the \texttt{JSIGNSET} structure.
Output
If successful, IERROR is set to zero; non-zero is set for a non-empty set.

**PXFSIGFILLSET**
*Fill a signal set.*

**Prototype**
```
INTERFACE
   SUBROUTINE PXFSIGFILLSET (JSIGSET, IERROR)
   INTEGER(4) JSIGSET, IERROR
   END SUBROUTINE PXFSIGFILLSET
END INTERFACE
```

JSIGNET  A set representing a group of signals (obtained with PXFSTRUCTURECREATE).
IERROR   Return code.

**Description**
This subroutine adds a signal to the JSIGNSET. It is useful for initializing a set.

**NOTE.** You must use the PXFSTRUCTURECREATE subroutine to create the JSIGNSET structure.

**Output**
If successful, IERROR is set to zero.
PXFSIGISMEMBER

*Checks if signal is a member of a set*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFSIGISMEMBER (JSIGSET, ISIGNO, ISMEMBER, IERROR)
    INTEGER(4) JSIGSET, ISIGNO, IERROR
    LOGICAL(4) ISMEMBER
  END SUBROUTINE PXFSIGISMEMBER
END INTERFACE
```

**JSIGNET**  
Input. A set representing a group of signals (obtained with PXFSTRUCTURECREATE).

**ISIGNO**  
Input. Signal to be tested for membership.

**ISMEMBER**  
Output. Logical result.

**IERROR**  
Output. Return code.

**Description**

This subroutine checks if the specified signal is a member of the specified set.

**Output**

If true, ISMEMBER contains .TRUE.. Otherwise, it is set to .FALSE..
PXFSTAT

*Gets the status of a file*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFSTAT (PATH, ILEN, JSTAT, IERROR)
  CHARACTER(LEN=*) PATH
  INTEGER(4) ILEN, JSTAT, IERROR
  END SUBROUTINE PXFSTAT
END INTERFACE
```

**PATH**  
Input. File name.

**ILEN**  
Input. Length of PATH string.

**JSTAT**  
Input. Pointer to PXFSTAT structure.

**IERROR**  
Output. Return value.

**Description**

This subroutine gets the status of a file.

**Output**

If successful, IERROR is set to zero.
PXFSTRGET

*Gets an integer component of a structure*

**Prototype**

```
INTERFACE
  SUBROUTINE PXFSTRGET (JHANDLE, COMPNAME, INDEX, VALUE, ILEN, IERROR)
    INTEGER(4)  JHANDLE, INDEX, ILEN, IERROR
    CHARACTER(LEN=*) COMPNAME, VALUE
  END SUBROUTINE PXFSTRGET
END INTERFACE
```

**JHANDLE**
Handle to the structure.

**COMPNAME**
Component name.

**INDEX**
Index of the component.

**VALUE**
Output. Where the value of the component is stored.

**ILEN**
Output. Length of VALUE string.

**IERROR**
Return value.

**Description**

This subroutine gets an integer component of a structure.

**Output**

If successful, IERROR is set to zero.
PXFSTRUCTCOPY

Copies the contents of one structure to another

Prototype

INTERFACE
  SUBROUTINE PXFSTRUCTCOPY (STRUCTNAME, JHANDLE1, JHANDLE2, IERROR)
  INTEGER(4) JHANDLE1, JHANDLE2, IERROR
  CHARACTER(LEN=*) STRUCTNAME
  END SUBROUTINE PXFSTRUCTCOPY
END INTERFACE

STRUCTNAME    Input. Structure name.
JHANDLE1      Input. Handle to the structure to be copied.
JHANDLE2      Input. Handle to structure to be copied into.
IERROR        Output. Return value.

Description

This subroutine copies the contents of one structure to another.

Output

If successful, IERROR is set to zero.
PXFSTRUCTCREATE

Creates an instance of the specified structure

Prototype

INTERFACE
  SUBROUTINE PXFSTRUCTCREATE (STRUCTNAME, JHANDLE, IERROR)
    CHARACTER(LEN=*) STRUCTNAME
    INTEGER(4) JHANDLE, IERROR
  END SUBROUTINE PXFSTRUCTCREATE
END INTERFACE

STRUCTNAME       Input. Structure name.
JHANDLE          Output. Handle to structure.
IERROR           Output. Return value.

Description

This subroutine creates an instance of the specified structure.

Output

If successful, IERROR is set to zero.
PXFSTRUCTFREE

Deletes the instance of a structure

Prototype

INTERFACE
SUBROUTINE PXFSTRUCTFREE (JHANDLE, IERROR)
INTEGER(4) JHANDLE, IERROR
END SUBROUTINE PXFSTRUCTFREE
END INTERFACE

JHANDLE Input. Handle to structure.
IERROR Output. Return value.

Description
This subroutine deletes the instance of a structure.

Output
If successful, IERROR is set to zero.

PXFUCOMPARE

Compares two unsigned numbers and returns the absolute value of their difference

Prototype

INTERFACE
SUBROUTINE PXFUCOMPARE (I1, I2, ICMPR, IDIFF)
INTEGER(4) I1, I2, ICMPR, IDIFF
END SUBROUTINE PXFUCOMPARE
END INTERFACE
I1, I2 The two unsigned integers to compare.
ICMPR Output. Result of comparison (-1,0,1).
IDIFF Output. Absolute value of the difference.

Description
This subroutine compares two unsigned numbers and returns the absolute value of their difference.

Output
The value of ICMPR upon completion:
-1 if I1 < I2
0 if I1 = I2
1 if I1 > I2

PXFUMASK
Sets and gets the file creation mask

Prototype
INTERFACE
   SUBROUTINE PXFUMASK (ICMASK, IPREVCMASK, IERROR)
   INTEGER(4) ICMASK, IPREVCMASK, IERROR
END SUBROUTINE PXFUMASK
END INTERFACE

ICMASK Input. Set mask.
IPREVCMASK Output. Previous mask.
IERROR Output. Return value.

Description
This subroutine sets and gets the file creation mask.
Output

If successful, IERROR is set to zero.

**PXFUNLINK**

*Removes the specified directory entry*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PXFUNLINK (PATH, ILEN, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, IERROR
  END SUBROUTINE PXFUNLINK
END INTERFACE
```

- **PATH**
  - Input. Name of the directory entry.
- **ILEN**
  - Input. Length of PATH string.
- **IERROR**
  - Output. Return value.

**Description**

This subroutine removes the specified directory entry.

**Output**

If successful, IERROR is set to zero.
PXFUTIME

Sets file access and modification times

Prototype

INTERFACE
  SUBROUTINE PXFUTIME (PATH, ILEN, JUTIMBUG, IERROR)
    CHARACTER(LEN=*) PATH
    INTEGER(4) ILEN, JUTIMBUF, IERROR
  END SUBROUTINE PXFUTIME
END INTERFACE

PATH      Input. File name.
ILEN      Input. Length of PATH string.
JUTIMBUG  Input. Pointer to PXFUTIMBUF structure.
IERROR    Output. Return value.

Description

This subroutine sets the file access and modification times.

Output

If successful, IERROR is set to zero.
PXFWAIT

Waits for any child process

Prototype

INTERFACE

SUBROUTINE PXFWAIT (ISTATE, IRETPID, IERROR)
INTEGER(4) ISTAT, IRETPID, IERROR
END SUBROUTINE PXFWAIT

END INTERFACE

ISTATE  Output. Status of child process.
IRETPID Input/output. The process ID to wait on when the
         function is called; the process ID of the stopped child
         process returned.
IERROR  Output. Error value.

Description

This subroutine waits for any child process.

Output

If successful, IERROR is set to zero.

PXFWAITPID

Waits for certain PIDs

Prototype

INTERFACE

SUBROUTINE PXFWAITPID (IPID, ISTAT, IOPTIONS, &
                         IRETPID, IERROR)
INTEGER(4) IPID, ISTAT, IOPTIONS, IRETPID, IERROR
END SUBROUTINE PXFWAITPID
END INTERFACE

IPID     Input. specifies the PIDs to wait for.
ISTAT    Output. status of child process.
IOPTIONS Input. On Linux, check.
           /usr/include/sys/wait.h for possible values of
           the options. WCONTINUED, WNOHANG, WNOWAIT,
           WUNTRACED. On Win32 systems, check your MSVC
           installation, in the include\process.h file.
IRETPID  Output. pid of the stopped child.
IERROR   Output. error value.

Description
This subroutine waits for certain PIDs.

Output
The PID and status of the child process that triggered the WAITPID
routine to stop waiting.

PXFWIFEXITED
Determines if a child process has exited.

Prototype
INTERFACE
   LOGICAL(4) FUNCTION PXFWIFEXITED (ISTAT)
   INTEGER(4) ISTAT
END FUNCTION PXFWIFEXITED
END INTERFACE

ISTAT     (output) an integer status obtained from
           PXFWEXITSTATUS.
Description
Tests whether a child process has exited.

Output
The function returns .TRUE. if ISTAT is equal to zero, and .FALSE. if ISTAT is not zero.

PXFWIFSIGNALED
Determines if a child process has exited due to a signal

Prototype
INTERFACE
   LOGICAL(4) FUNCTION PXFWIFSIGNALED (ISTAT)
   INTEGER(4) ISTAT
   END FUNCTION PXFWIFSIGNALED
END INTERFACE

ISTAT
On Win32 systems, check your MSVC installation, in the include\process.h file. (output) the status variable returned from a previous call to PXFWAIT or PXFWAITPID for a child process.

Description
PXFWIFSIGNALED evaluates the status code returned from a previous call to PXFWAITPID for a child process. If the child terminated due to an missed signal, PXFWIFSIGNALED returns .TRUE..

Output
The function returns .TRUE. if ISTAT is equal to zero, and .FALSE. if ISTAT is not zero.
PXFWIFSTOPPED

Determines if a child process is currently stopped or suspended

Prototype

INTERFACE
   LOGICAL(4) FUNCTION PXFWIFSTOPPED (ISTAT)
   INTEGER(4) ISTAT
   END FUNCTION PXFWIFSTOPPED
END INTERFACE

ISTAT (output) the status variable returned from a previous call to PXFWAIT or PXFWAITPID for a child process

Description

PXFWIFSTOPPED evaluates the input status code to determine whether a particular child process is currently stopped.

Output

The function returns .TRUE. if ISTAT is equal to zero, and .FALSE. if ISTAT is not zero.

PXWRITE

Writes to a file

Prototype

INTERFACE
   SUBROUTINE PXWRITE (IFILDES, BUF, NBYTE, &
                        NWRITTEN, IERROR)

INTEGER(4) IFILDES, NBYTE, NWRITE, IERROR
CHARACTER BUF(LEN=*)
END SUBROUTINE PXfwrite
END INTERFACE

IFILDES       Input. Descriptor to be written to.
BUF           Input. Buffer to be written from.
NBYTE         Input. Number of bytes to write.
NWRITE        Output. Number of bytes written.
IERROR        Output. Returned error code.

Description
This subroutine writes output to a file.

Output
If successful, IERROR is set to zero.
This chapter describes procedures that comprise the QuickWin library (libqwin.lib). These procedures are made available to the compiler when you invoke the \MW option and use the flib.fd include file.

The prototypes are described using the INTERFACE call, which provides the required information to complete a call to the specified procedure. For descriptions of the INTERFACE block and the \MW option, see the Intel® Fortran Compiler User's Guide.

You can use the QuickWin library to compile programs into simple applications for Windows* which allow you to call graphic routines, load and save bitmaps, perform edit and other operations available under windows.

To use the QuickWin library, your program must explicitly access this library procedures with this statement:

```
INCLUDE "<installation directory>\include\flib.fd"
```

If a procedure does not have a PROGRAM statement, then the above statement must appear in each subprogram that makes graphics calls.

This chapter introduces the major groups of QuickWin library procedures described in the following sections:

- QuickWin subroutines and functions
- Graphics subroutines and functions
**QuickWin Subroutines and Functions**

This category includes subroutines and functions that provide windows control and configuration, enhance QuickWin applications and perform color conversion. To use the procedures of this group, add the following statement to the program unit containing the procedure:

```
INCLUDE "<installation directory>\include\flib.fd"
```

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Windows Control</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOCUSQQ</td>
<td>Function, result = FOCUSQQ (iunit)</td>
<td>Sets focus to specified window.</td>
</tr>
<tr>
<td>GETACTIVEQQ</td>
<td>Function, result = GETACTIVEQQ ( )</td>
<td>Returns the unit number of the currently active child.</td>
</tr>
<tr>
<td>GETHWNDQQ</td>
<td>Function, result = GETHWNDQQ (unit)</td>
<td>Converts the unit number into a Windows handle for functions that require it.</td>
</tr>
<tr>
<td>GETWINDOWCONFIG</td>
<td>Function, result = GETWINDOWCONFIG (wc)</td>
<td>Returns current window properties.</td>
</tr>
<tr>
<td>GETWSIZEQQ</td>
<td>Function, result = GETWSIZEQQ (unit, ireq, winfo)</td>
<td>Returns the size and position of a window.</td>
</tr>
<tr>
<td>INQFOCUSQQ</td>
<td>Function, result = INQFOCUSQQ (unit)</td>
<td>Determines which window has focus.</td>
</tr>
<tr>
<td>SETACTIVEQQ</td>
<td>Function, result = SETACTIVEQQ (unit)</td>
<td>Makes a child window active, but does not give it focus.</td>
</tr>
<tr>
<td>SETWINDOWCONFIG</td>
<td>Function, result = SETWINDOWCONFIG (wc)</td>
<td>Sets current window properties.</td>
</tr>
<tr>
<td>SETWSIZEQQ</td>
<td>Function, result = SETWSIZEQQ (unit, winfo)</td>
<td>Sets the size and position of a window.</td>
</tr>
</tbody>
</table>

continued
### Table 4-1  QuickWin Routines (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QuickWin Applications Enhancements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABOUTBOXQQ</td>
<td>Function</td>
<td>Adds an About Box with customized text.</td>
</tr>
<tr>
<td>result = ABOUTBOXQQ (cstring)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APPENDMENUQQ</td>
<td>Function</td>
<td>Appends a menu item.</td>
</tr>
<tr>
<td>result = APPENDMENUQQ (menuID, flags, text, routine)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLICKMENUQQ</td>
<td>Function</td>
<td>Simulates the effect of clicking or selecting a menu item.</td>
</tr>
<tr>
<td>result = CLICKMENUQQ (item)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETEMENUQQ</td>
<td>Function</td>
<td>Deletes a menu item.</td>
</tr>
<tr>
<td>result = DELETEMENUQQ (menuID, itemID)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETEXITQQ</td>
<td>Function</td>
<td>Returns the setting for a QuickWin application's exit behavior.</td>
</tr>
<tr>
<td>result = GETEXITQQ ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCHARQQ</td>
<td>Function</td>
<td>Reads a single character input from the keyboard and returns the ASCII value of that character without any buffering.</td>
</tr>
<tr>
<td>result = INCHARQQ ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIALSETTINGS</td>
<td>Function</td>
<td>Controls initial menu settings and initial frame window.</td>
</tr>
<tr>
<td>result = INITIALSETTINGS ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERTMENUQQ</td>
<td>Function</td>
<td>Inserts a menu item.</td>
</tr>
<tr>
<td>result = INSERTMENUQQ (menuID, itemID, flag, text, routine)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSAGEBOXQQ</td>
<td>Function</td>
<td>Displays a message box.</td>
</tr>
<tr>
<td>result = MESSAGEBOXQQ (msg, caption, mtype)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODIFYMENUFLAGSQQ</td>
<td>Function</td>
<td>Modifies a menu item's state.</td>
</tr>
<tr>
<td>result = MODIFYMENUFLAGSQQ (menuID, itemID, flag)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued
### Table 4-1 QuickWin Routines (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODIFYMENUROUTINEQQ</td>
<td>result = MODIFYMENUROUTINEQQ (menuID, itemID, routine)</td>
<td>Function</td>
</tr>
<tr>
<td>MODIFYMENSTRINGQQ</td>
<td>result = MODIFYMENSTRINGQQ (menuID, itemID, text)</td>
<td>Function</td>
</tr>
<tr>
<td>REGISTERMOUSEEVENT</td>
<td>result = REGISTERMOUSEEVENT (unit, mouseevents, callbackroutine)</td>
<td>Function</td>
</tr>
<tr>
<td>SETEXITQQ</td>
<td>result = SETEXITQQ (exitmode)</td>
<td>Function</td>
</tr>
<tr>
<td>SETMESSAGEQQ</td>
<td>CALL SETMESSAGEQQ (msg, id)</td>
<td>Subroutine</td>
</tr>
<tr>
<td>SETWINDOWMENUQQ</td>
<td>result = SETWINDOWMENUQQ (menuID)</td>
<td>Function</td>
</tr>
<tr>
<td>UNREGISTERMOUSEEVENT</td>
<td>result = UNREGISTERMOUSEEVENT (unit, mouseevents)</td>
<td>Function</td>
</tr>
<tr>
<td>WAITONMOUSEEVENT</td>
<td>result = WAITONMOUSEEVENT (mouseevents, keystate, x, y)</td>
<td>Function</td>
</tr>
</tbody>
</table>

**Color Conversion**

| INTEGERTORGB | CALL INTEGERTORGB (rgb, red, green, blue) | Subroutine | Converts an RGB color value to its red, green, and blue components |

---

**continue**
The QuickWin run-time library helps you turn graphics programs into simple Windows applications. QuickWin applications support pixel-based graphics, real-coordinate graphics, text windows, character fonts, user-defined menus, mouse events, and editing (select/copy/paste) of text, graphics, or both.

To use the QuickWin library, invoke the compiler driver with the option /MW to link to the QuickWin library, libqwin.lib, and add the following statement to the program unit containing the QuickWin procedure(s):

```
INCLUDE "<installation directory>\include\flib.fd"
```

When QuickWin is not being used, you link by default to the library libqwind.lib instead. This library consists of dummy versions of QuickWin functions to avoid unresolved references at link time.

Note that QuickWin applications cannot be DLLs and QuickWin cannot be linked with runtime routines that are in DLLs.

Table 4-2 summarizes the Graphics procedures. When writing your applications, you can use any case.

---

**Table 4-1 QuickWin Routines (continued)**

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGBTOINTEGER</td>
<td>Function</td>
<td>Converts integers specifying red, green, and blue color into an RGB integer (for use in RGB routines).</td>
</tr>
</tbody>
</table>

---

**Table 4-1 QuickWin Routines (continued)**

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGBTOINTEGER</td>
<td>Function</td>
<td>Converts integers specifying red, green, and blue color into an RGB integer (for use in RGB routines).</td>
</tr>
</tbody>
</table>

---

**Graphics Procedures**

The QuickWin run-time library helps you turn graphics programs into simple Windows applications. QuickWin applications support pixel-based graphics, real-coordinate graphics, text windows, character fonts, user-defined menus, mouse events, and editing (select/copy/paste) of text, graphics, or both.

To use the QuickWin library, invoke the compiler driver with the option /MW to link to the QuickWin library, libqwin.lib, and add the following statement to the program unit containing the QuickWin procedure(s):

```
INCLUDE "<installation directory>\include\flib.fd"
```

When QuickWin is not being used, you link by default to the library libqwind.lib instead. This library consists of dummy versions of QuickWin functions to avoid unresolved references at link time.

Note that QuickWin applications cannot be DLLs and QuickWin cannot be linked with runtime routines that are in DLLs.

Table 4-2 summarizes the Graphics procedures. When writing your applications, you can use any case.
### Table 4-2 Graphics Routines and Functions Summary

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARC, ARC_W</strong></td>
<td>Function</td>
<td>Draws an arc.</td>
</tr>
<tr>
<td>result = ARC (x1, y1, x2, y2, x3, y3, x4, y4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = ARC_W (wx1, wy1, wx2, wy2, wx3, wy3, wx4, wy4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CLEARSCREEN</strong></td>
<td>Subroutine</td>
<td>Clears the screen, viewport, or text window.</td>
</tr>
<tr>
<td>CALL CLEARSCREEN (area)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DISPLAYCURSOR</strong></td>
<td>Function</td>
<td>Graphics Function Turns the cursor off and on.</td>
</tr>
<tr>
<td>result = DISPLAYCURSOR (toggle)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ELLIPSE, ELLIPSE_W</strong></td>
<td>Function</td>
<td>Draws an ellipse or circle</td>
</tr>
<tr>
<td>result = ELLIPSE (control, x1, y1, x2, y2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = ELLIPSE_W (control, wx1, wy1, wx2, wy2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FLOODFILL, FLOODFILL_W</strong></td>
<td>Function</td>
<td>Fills an enclosed area of the screen with the current color index, using the current fill mask.</td>
</tr>
<tr>
<td>result = FLOODFILL (x, y, bcolor)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = FLOODFILL_W (wx, wy, bcolor)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FLOODFILLRGB, FLOODFILLRGB_W</strong></td>
<td>Function</td>
<td>Fills an enclosed area of the screen with the current RGB color, using the current.</td>
</tr>
<tr>
<td>result = FLOODFILLRGB (x, y, color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = FLOODFILLRGB_W (wx, wy, color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GETARCINFO</strong></td>
<td>Function</td>
<td>Determines the end points of the most recently drawn arc or pie.</td>
</tr>
<tr>
<td>result = GETARCINFO (pstart, pend, ppaint)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GETBKCOLOR</strong></td>
<td>Function</td>
<td>Returns the current background color index.</td>
</tr>
<tr>
<td>result = GETBKCOLOR ( )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4-2  Graphics Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETBKCOLORRGB</td>
<td>Function</td>
<td>Returns the current background RGB color.</td>
</tr>
<tr>
<td>result = GETBKCOLORRGB ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETCOLOR</td>
<td>Function</td>
<td>Returns the current color index.</td>
</tr>
<tr>
<td>result = GETCOLOR ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETCOLORRGB</td>
<td>Function</td>
<td>Returns the current RGB color.</td>
</tr>
<tr>
<td>result = GETCOLORRGB ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETCURRENTPOSITION,</td>
<td>Subroutine</td>
<td>Returns the coordinates of the current graphics-output position.</td>
</tr>
<tr>
<td>GETCURRENTPOSITION_W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL GETCURRENTPOSITION (t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL GETCURRENTPOSITION_W (wt)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETFILLMASK</td>
<td>Subroutine</td>
<td>Returns the current fill mask.</td>
</tr>
<tr>
<td>CALL GETFILLMASK (mask)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETFONTINFO</td>
<td>Function</td>
<td>Returns the current font characteristics.</td>
</tr>
<tr>
<td>result = GETFONTINFO (font)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETGTEXTENT</td>
<td>Function</td>
<td>Determines the width of the specified text in the current font.</td>
</tr>
<tr>
<td>result = GETGTEXTENTENT (text)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETGTEXTROTATION</td>
<td>Function</td>
<td>Get the current text rotation angle.</td>
</tr>
<tr>
<td>result = GETGTEXTROTATION ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETIMAGE, GETIMAGE_W</td>
<td>Subroutine</td>
<td>Stores a screen image in memory.</td>
</tr>
<tr>
<td>CALL GETIMAGE (x1, y1, x2, y2, image)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL GETIMAGE_W (wx1, wy1, wx2, wy2, image)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETLINESTYLE</td>
<td>Function</td>
<td>Returns the current line style.</td>
</tr>
<tr>
<td>result = GETLINESTYLE ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETPHYSCOORD</td>
<td>Subroutine</td>
<td>Converts viewport coordinates to physical coordinates.</td>
</tr>
<tr>
<td>CALL GETPHYSCOORD (x, y, t)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued
Table 4-2  Graphics Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPIXEL, GETPIXEL_W</td>
<td>Function</td>
<td>Returns a pixel's color index.</td>
</tr>
</tbody>
</table>
| result = GETPIXEL (x, y)  
result = GETPIXEL_W (wx, wy) | |
| GETPIXELRGB, GETPIXELRGB_W | Function | Returns a pixel's RGB color. |
| result = GETPIXELRGB (x, y)  
result = GETPIXELRGB_W (wx, wy) | |
| GETPIXELS | Function | Returns the color indices of multiple pixels. |
| CALL GETPIXELS (n, x, y, color) | |
| GETPIXELSRGB | Function | Returns the RGB colors of multiple pixels. |
| CALL GETPIXELSRGB (n, x, y, color) | |
| GETTEXTCOLOR | Function | Returns the current text color index |
| result = GETTEXTCOLOR () | |
| GETTEXTCOLORRGB | Function | Returns the current text RGB color. |
| result = GETTEXTCOLORRGB () | |
| GETTEXTPOSITION | Subroutine | Returns the current text-output position. |
| CALL GETTEXTPOSITION (t) | |
| GETTEXTWINDOW | Subroutine | Returns the boundaries of the current text window. |
| CALL GETTEXTWINDOW (rl, cl, r2, c2) | |
| GETVIEWCOORD, GETVIEWCOORD_W | Subroutine | Converts physical or window coordinates to viewport coordinates. |
| CALL GETVIEWCOORD (x, y, t)  
CALL GETVIEWCOORD_W (wx, wy, wt) | |
| GETWINDOWCOORD | Subroutine | Converts viewport coordinates to window coordinates. |
| CALL GETWINDOWCOORD (x, y, wt) | |
| GETWRITEMODE | Function | Returns the logical write mode for lines. |
| result = GETWRITEMODE () | |

continued
<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GRSTATUS</strong></td>
<td>Function</td>
<td>Returns the status (success or failure) of the most recently called graphics routine.</td>
</tr>
<tr>
<td>result = GRSTATUS ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IMAGESIZE, IMAGESIZE_W</strong></td>
<td>Function</td>
<td>Returns image size in bytes.</td>
</tr>
<tr>
<td>result = IMAGESIZE (x1, y1, x2, y2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = IMAGESIZE_W (wx1, wy1, wx2, wy2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INITIALIZEFONTS</strong></td>
<td>Function</td>
<td>Initializes the font library.</td>
</tr>
<tr>
<td>result = INITIALIZEFONTS ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LINETO, LINETO_W</strong></td>
<td>Function</td>
<td>Draws a line from the current position to a specified point.</td>
</tr>
<tr>
<td>result = LINETO (x, y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = LINETO_W (wx, wy)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LINETOAR</strong></td>
<td>Function</td>
<td>Draws a line between points in one array and corresponding points in another array.</td>
</tr>
<tr>
<td>result = LINETOAR (loc(fx), loc(fy), loc(tx), loc(ty), cnt)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LINETOAREX</strong></td>
<td>Function</td>
<td>Similar to LINETOAR, and in addition specifies the color and line style.</td>
</tr>
<tr>
<td>result = LINETOAR (loc(fx), loc(fy), loc(tx), loc(ty), loc(C), loc(S), cnt)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LOADIMAGE, LOADIMAGE_W</strong></td>
<td>Function</td>
<td>Reads a Windows bitmap file (.BMP) and displays it at the specified location.</td>
</tr>
<tr>
<td>result = LOADIMAGE (filename, xcoord, ycoord)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = LOADIMAGE_W (filename, wxcoord, wycoord)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MOVETO, MOVETO_W</strong></td>
<td>Subroutine</td>
<td>Moves the current position to the specified point.</td>
</tr>
<tr>
<td>CALL MOVETO (x, y, t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL MOVETO_W (wx, wy, w)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4-2 Graphics Routines and Functions Summary (continued)
### Table 4-2  Graphics Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OUTGTEXT</strong></td>
<td>Subroutine</td>
<td>Sends text in the current font to the screen at the current position.</td>
</tr>
<tr>
<td>CALL OUTGTEXT (text)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OUTTEXT</strong></td>
<td>Subroutine</td>
<td>Sends text to the screen at the current position.</td>
</tr>
<tr>
<td>CALL OUTTEXT (text)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PIE, PIE_W</strong></td>
<td>Function</td>
<td>Draws a pie slice.</td>
</tr>
<tr>
<td>result = PIE (i, x1, y1, x2, y2, x3, y3, x4, y4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = PIE_W (i, wx1, wy1, wx2, wy2, wx3, wy3, wx4, wy4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>POLYGON, POLYGON_W</strong></td>
<td>Function</td>
<td>Draws a polygon.</td>
</tr>
<tr>
<td>result = POLYGON (control, ppoints, cpoints)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = POLYGON_W (control, wppoints, cpoints)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PUTIMAGE, PUTIMAGE_W</strong></td>
<td>Subroutine</td>
<td>Retrieves an image from memory and displays it.</td>
</tr>
<tr>
<td>CALL PUTIMAGE (x, y, image, action)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL PUTIMAGE_W (wx, wy, image, action)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RECTANGLE, RECTANGLE_W</strong></td>
<td>Function</td>
<td>Draws a rectangle.</td>
</tr>
<tr>
<td>result = RECTANGLE (control, x1, y1, x2, y2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = RECTANGLE_W (control, wx1, wy1, wx2, wy2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REMAPALLPALETTERGB</strong></td>
<td>Function</td>
<td>Remaps a set of RGB color values to indices recognized by the current video configuration.</td>
</tr>
<tr>
<td>result = REMAPALLPALETTERGB (colors)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REMAPPALETTERGB</strong></td>
<td>Function</td>
<td>Remaps a single RGB color value to a color index.</td>
</tr>
<tr>
<td>result = REMAPPALETTERGB (index, color)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued
Table 4-2  Graphics Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAVEIMAGE, SAVEIMAGE_W</strong></td>
<td>Function</td>
<td>Captures a screen image and saves it as a Windows bitmap file.</td>
</tr>
<tr>
<td>result = SAVEIMAGE (filename, ulxcoord, ulycoord, lrxcoord, lrycoord)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = SAVEIMAGE_W (filename, ulwxcoord, ulwycoord, lrwxcoord, lrwycoord)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SAVEJPEG, SAVEJPEG_W</strong></td>
<td>Function</td>
<td>Captures a screen image and saves it into a JPEG graphic file.</td>
</tr>
<tr>
<td>result = SAVEJPEG (filename, ulxcoord, ulycoord, lrxcoord, lrycoord, jpgquality)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = SAVEJPEG_W (filename, ulwxcoord, ulwycoord, lrwxcoord, lrwycoord, jpgwquality)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SCROLLTEXTWINDOW</strong></td>
<td>Function</td>
<td>Scrolls the contents of a text window.</td>
</tr>
<tr>
<td>CALL SCROLLTEXTWINDOW (rows)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SETBKCOLOR</strong></td>
<td>Function</td>
<td>Sets the current background color.</td>
</tr>
<tr>
<td>result = SETBKCOLOR (color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SETBKCOLORRGB</strong></td>
<td>Function</td>
<td>Sets the current background color to a direct color value rather than an index to a defined palette.</td>
</tr>
<tr>
<td>result = SETBKCOLORRGB (color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SETCLIPRGN</strong></td>
<td>Subroutine</td>
<td>Limits graphics output to a part of the screen.</td>
</tr>
<tr>
<td>CALL SETCLIPRGN (x1, y1, x2, x2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SETCOLOR</strong></td>
<td>Function</td>
<td>Sets the current color to a new color index.</td>
</tr>
<tr>
<td>result = SETCOLOR (color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SETCOLORRGB</strong></td>
<td>Function</td>
<td>Sets the current color to a direct color value rather than an index to a defined palette.</td>
</tr>
<tr>
<td>result = SETCOLORRGB (color)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued
Table 4-2  Graphics Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETFILLMASK</td>
<td>Subroutine</td>
<td>Changes the current fill mask to a new pattern.</td>
</tr>
<tr>
<td>CALL SETFILLMASK (mask)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETFONT</td>
<td>Function</td>
<td>Finds a single font matching the specified characteristics and assigns it to OUTGTEXT.</td>
</tr>
<tr>
<td>result = SETFONT (options)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETGTEXTROTATION</td>
<td>Subroutine</td>
<td>Sets the direction in which text is written to the specified angle.</td>
</tr>
<tr>
<td>CALL SETGTEXTROTATION (degrees)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETLINESTYLE</td>
<td>Subroutine</td>
<td>Changes the current line style.</td>
</tr>
<tr>
<td>CALL SETLINESTYLE (mask)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETPIXEL, SETPIXEL_W</td>
<td>Function</td>
<td>Sets color of a pixel at a specified location.</td>
</tr>
<tr>
<td>result = SETPIXEL (x, y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = SETPIXEL_W (wx, wy)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETPIXELRGB, SETPIXELRGB_W</td>
<td>Function</td>
<td>Sets RGB color of a pixel at a specified location.</td>
</tr>
<tr>
<td>result = SETPIXELRGB (x, y, color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result = SETPIXELRGB_W (wx, wy, color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETPIXELS</td>
<td>Subroutine</td>
<td>Sets the color indices of multiple pixels.</td>
</tr>
<tr>
<td>CALL SETPIXELS (n, x, y, color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETPIXELSRGB</td>
<td>Subroutine</td>
<td>Sets the RGB color of multiple pixels.</td>
</tr>
<tr>
<td>CALL SETPIXELSRGB (n, x, y, color)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETTEXTCOLOR</td>
<td>Function</td>
<td>Sets the current text color to a new color index.</td>
</tr>
<tr>
<td>result = SETTEXTCOLOR (index)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETTEXTCOLORRGB</td>
<td>Function</td>
<td>Sets the current text color to a direct color value rather than an index to a defined palette.</td>
</tr>
<tr>
<td>result = SETTEXTCOLORRGB (color)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued
### Table 4-2 Graphics Routines and Functions Summary (continued)

<table>
<thead>
<tr>
<th>Name/Syntax</th>
<th>Subroutine / Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETTEXTPOSITION</td>
<td>Subroutine</td>
<td>Changes the current text position.</td>
</tr>
<tr>
<td>CALL SETTEXTPOSITION (row, column, t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETTEXTWINDOW</td>
<td>Subroutine</td>
<td>Sets the current text display window.</td>
</tr>
<tr>
<td>CALL SETTEXTWINDOW (r1, c1, r2, c2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETVIEWORG</td>
<td>Subroutine</td>
<td>Positions the viewport coordinate origin.</td>
</tr>
<tr>
<td>CALL SETVIEWORG (x, y, t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETVIEWPORT</td>
<td>Subroutine</td>
<td>Defines the size and screen position of the viewport.</td>
</tr>
<tr>
<td>CALL SETVIEWPORT (x1, y1, x2, y2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETWINDOW</td>
<td>Function</td>
<td>Defines the window coordinate system.</td>
</tr>
<tr>
<td>result = SETWINDOW (finvert, wx1, wy1, wx2, wy2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETWRITEMODE</td>
<td>Function</td>
<td>Changes the current logical write mode for lines.</td>
</tr>
<tr>
<td>result = SETWRITEMODE (wmode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRAPON</td>
<td>Function</td>
<td>Turns line wrapping on or off.</td>
</tr>
<tr>
<td>result = WRAPON (option)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graphic Function Descriptions

ARC

*Draws elliptical arcs using the current graphics color*

**Prototype**

```fortran
INTERFACE
  FUNCTION ARC(X1,Y1,X2,Y2,X3,Y3,X4,Y4)
    INTEGER(2) ARC,X1,Y1,X2,Y2,X3,Y3,X4,Y4
  END FUNCTION
END INTERFACE
```

**Description**

This function draws elliptical arcs using the current graphics color. The center of the arc is the center of the bounding rectangle defined by the points \((X1, Y1)\) and \((X2, Y2)\).

The arc starts where it intersects an imaginary line extending from the center of the arc through \((X3, Y3)\). It is drawn counterclockwise about the center of the arc, ending where it intersects an imaginary line extending from the center of the arc through \((X4, Y4)\).

ARC uses the view-coordinate system. The arc is drawn using the current color.
Output
The result type is `INTEGER(2)`. It is nonzero if successful; otherwise, 0. If the arc is clipped or partially out of bounds, the arc is considered successfully drawn and the return is 1. If the arc is drawn completely out of bounds, the return is 0.

---

**ARC_W**

*Draws elliptical arcs using the current graphics color*

**Prototype**

```fortran
INTERFACE
  FUNCTION ARC_W(WX1,WY1,WX2,WY2,WX3,WY3,WX4,WY4)
    INTEGER(2) ARC_W
    DOUBLE PRECISION WX1,WY1,WX2,WY2,WX3,WY3,WX4,WY4
  END FUNCTION
END INTERFACE
```

**Description**

This function draws elliptical arcs using the current graphics color. The center of the arc is the center of the bounding rectangle defined by the points `(WX1, WY1)` and `(WX2, WY2)`. 

**Input**

- `WX1, WY1` : Input. REAL(8). Window coordinates for upper-left corner of bounding rectangle.
- `WX2, WY2` : Input. REAL(8). Window coordinates for lower-right corner of bounding rectangle.
- `WX3, WY3` : Input. REAL(8). Window coordinates of start vector.
- `WX4, WY4` : Input. REAL(8). Window coordinates of end vector.
The arc starts where it intersects an imaginary line extending from the center of the arc through \((WX3, WY3)\). It is drawn counterclockwise about the center of the arc, ending where it intersects an imaginary line extending from the center of the arc through \((WX4, WY4)\).

\(ARC\_W\) uses the view-coordinate system. The arc is drawn using the current color.

**Output**

The result type is \texttt{INTEGER(2)}. It is nonzero if successful; otherwise, 0. If the arc is clipped or partially out of bounds, the arc is considered successfully drawn and the return is 1. If the arc is drawn completely out of bounds, the return is 0.

---

**CLEARSCREEN**

*Erases the target area and fills it with the current background color*

**Prototype**

```
INTERFACE
    SUBROUTINE CLEARSCREEN (AREA)
    INTEGER(2) AREA
    END SUBROUTINE
END INTERFACE
```

**AREA**

Input. \texttt{INTEGER(4)}. Identifies the target area.

**Description**

Parameter \texttt{AREA} identifies the target area. Must be one of the following symbolic constants (defined in \texttt{flib.fd}):

- \$\texttt{GCLEARSCREEN}  
  Clears the entire screen.
- \$\texttt{GVVIEWPORT}  
  Clears only the current viewport.
QuickWinLibrary

$GWINDOW
Clears only the current text window (set with SETTEXTWINDOW).
All pixels in the target area are set to the color specified with SETBKCOLORRGB. The default color is black.

Output
None

DISPLAYCURSOR
Controls cursor visibility

Prototype
INTERFACE
FUNCTION DISPLAYCURSOR(TOGGLE)
INTEGER(2) DISPLAYCURSOR, TOGGLE
END FUNCTION
END INTERFACE

TOGGLE
Input. INTEGER (2): Constant that defines the cursor state. Has two values:

$GCURSOROFF
Makes the cursor invisible regardless of its current shape and mode.

$GCURSORON
Makes the cursor always visible in graphics mode.

Description
The function controls cursor visibility. Has these two values:
Cursor settings hold only for the currently active child window. You need to call DISPLAYCURSOR for each window in which you want the cursor to be visible.

A call to SETWINDOWCONFIG turns off the cursor.

**Output**

The result is the previous value of TOGGLE.

---

**ELLIPSE**

*Draws a circle or an ellipse using the current graphics color.*

**Prototype**

```fortran
INTERFACE
  FUNCTION ELLIPSE(CONTROL, X1, Y1, X2, Y2)
    INTEGER(2) ELLIPSE, CONTROL, X1, Y1, X2, Y2
  END FUNCTION
END INTERFACE
```

- **CONTROL**
  - Input. `INTEGER(2)`. Fill flag. Can be one of the following symbolic constants:
    - `$GFILLINTERIOR` Fills the figure using the current color and fill mask.
    - `$GBORDER` Does not fill the figure.

- **X1, Y1**
  - Input. `INTEGER(2)`. Viewport coordinates for upper-left corner of bounding rectangle.

- **X2, Y2**
  - Input. `INTEGER(2)`. Viewport coordinates for lower-right corner of bounding rectangle.
Description
When you use ELLIPSE, the center of the ellipse is the center of the bounding rectangle defined by the viewport-coordinate points \((x_1, y_1)\) and \((x_2, y_2)\). If the bounding-rectangle arguments define a point or a vertical or horizontal line, no figure is drawn.

The control option given by SGFILLINTERIOR is equivalent to a subsequent call to the FLOODFILLRGB function using the center of the ellipse as the start point and the current color (set by SETCOLORRGB) as the boundary color. The border is drawn in the current color and line style.

Output
The result type is INTEGER(2). The result is nonzero if successful; otherwise, 0. If the ellipse is clipped or partially out of bounds, the ellipse is considered successfully drawn, and the return is 1. If the ellipse is drawn completely out of bounds, the return is 0.

**ELLIPSE_W**

*Draws a circle or an ellipse using the current graphics color.*

**Prototype**

```fortran
INTERFACE
  FUNCTION ELLIPSE_W(CONTROL, WX1, WY1, WX2, WY2)
    INTEGER(2) ELLIPSE_W, CONTROL
    DOUBLE PRECISION WX1, WY1, WX2, WY2
  END FUNCTION
END INTERFACE
```

CONTROL  
*Input. INTEGER(2). Fill flag. Can be one of the following symbolic constants:*
$GFILLINTERIOR
Fills the figure using the current color and fill mask.

$GBORDER
Does not fill the figure.

WX1, WY1 Input. REAL(8). Window coordinates for upper-left corner of bounding rectangle.

WX2, WY2 Input. REAL(8). Window coordinates for lower-right corner of bounding rectangle.

Description
When you use ELLIPSE_W, the center of the ellipse is the center of the bounding rectangle defined by the window-coordinate points (WX1, WY1) and (WX2, WY2). If the bounding-rectangle arguments define a point or a vertical or horizontal line, no figure is drawn.

The control option given by $GFILLINTERIOR is equivalent to a subsequent call to the FLOODFILLRGB function using the center of the ellipse as the start point and the current color (set by SETCOLORRGB) as the boundary color.

The border is drawn in the current color and line style.

Output
The result type is INTEGER(2). The result is nonzero if successful; otherwise, 0. If the ellipse is clipped or partially out of bounds, the ellipse is considered successfully drawn, and the return is 1. If the ellipse is drawn completely out of bounds, the return is 0.
**FLOODFILL**

Fills an area using the current color index and fill mask.

---

**Prototype**

```fortran
INTERFACE
  FUNCTION FLOODFILL(X,Y,BOUNDARY)
    INTEGER(2) FLOODFILL, X, Y, BOUNDARY
  END FUNCTION
END INTERFACE

X, Y    Input. INTEGER(2). Viewport coordinates for fill starting point.
BCOLOR Input. INTEGER(2). Color index of the boundary color.
```

**Description**

FLOODFILL begins filling at the viewport-coordinate point \((X, Y)\). The fill color used by FLOODFILL is set by SETCOLOR. You can obtain the current fill color index by calling GETCOLOR. These functions allow access only to the colors in the palette (256 or less). To access all available colors on a VGA (262,144 colors) or a true color system, use the RGB functions FLOODFILLRGB and FLOODFILLRGB_W.

If the starting point lies inside a figure, the interior is filled; if it lies outside a figure, the background is filled. In both cases, the fill color is the current graphics color index set by SETCOLOR. The starting point must be inside or outside the figure, not on the figure boundary itself. Filling occurs in all directions, stopping at pixels of the boundary color BCOLOR.
Output

The result type is INTEGER(2). The result is a nonzero value if successful; otherwise, 0 (occurs if the fill could not be completed, or if the starting point lies on a pixel with the boundary color BCOLOR, or if the starting point lies outside the clipping region).

FLOODFILL_W

Fills an area using the current color index and fill mask.

Prototype

INTERFACE
  FUNCTION FLOODFILL_W(WX1, WY1, BOUNDARY)
    INTEGER(2) FLOODFILL_W, BOUNDARY
    DOUBLE PRECISION WX1, WY1
  END FUNCTION
END INTERFACE

WX1, WY1     Input. REAL(8). Window coordinates for fill starting point.
BCOLOR       Input. INTEGER(2). Color index of the boundary color.

Description

FLOODFILL_W begins filling at the window-coordinate point (WX1, WY1). The fill color used by FLOODFILL_W is set by SETCOLOR. You can obtain the current fill color index by calling GETCOLOR. These functions allow access only to the colors in the palette (256 or less). To access all available colors on a VGA (262,144 colors) or a true color system, use the RGB functions FLOODFILLRGB and FLOODFILLRGB_W.
If the starting point lies inside a figure, the interior is filled; if it lies outside a figure, the background is filled. In both cases, the fill color is the current graphics color index set by SETCOLOR. The starting point must be inside or outside the figure, not on the figure boundary itself. Filling occurs in all directions, stopping at pixels of the boundary color BCOLOR.

**Output**
The result type is INTEGER (2). The result is a nonzero value if successful; otherwise, 0 (occurs if the fill could not be completed, or if the starting point lies on a pixel with the boundary color BCOLOR, or if the starting point lies outside the clipping region).

**FLOODFILLRGB**
*Fills an area using the current Red-Green-Blue (RGB) color and fill mask.*

**Prototype**
```fortran
INTERFACE
  FUNCTION FLOODFILLRGB(X,Y,BCOLOR)
    INTEGER(2) FLOODFILLRGB,X,Y
    INTEGER(4) BCOLOR
  END FUNCTION
END INTERFACE
```

**X, Y**
Input INTEGER (2). Viewport coordinates for fill starting point.

**BCOLOR**
Input INTEGER (4). RGB value of the boundary color.
Description
FLOODFILLRGB begins filling at the viewport-coordinate point \((X, Y)\). The fill color used by FLOODFILLRGB is set by SETCOLORRGB. You can obtain the current fill color by calling GETCOLORRGB.

If the starting point lies inside a figure, the interior is filled; if it lies outside a figure, the background is filled. In both cases, the fill color is the current color set by SETCOLORRGB. The starting point must be inside or outside the figure, not on the figure boundary itself. Filling occurs in all directions, stopping at pixels of the boundary color \textit{color}.

Output
The result type is INTEGER(4). The result is a nonzero value if successful; otherwise, 0 (occurs if the fill could not be completed, or if the starting point lies on a pixel with the boundary color \textit{BCOLOR}, or if the starting point lies outside the clipping region).

**FLOODFILLRGB_W**
\textit{Fills an area using the current Red-Green-Blue (RGB) color and fill mask.}

Prototype
\begin{verbatim}
INTERFACE
  FUNCTION FLOODFILLRGB_W(WX,WY,BCOLOR)
    INTEGER(2) FLOODFILLRGB_W
    DOUBLE PRECISION WX,WY
    INTEGER(4) BCOLOR
  END FUNCTION
END INTERFACE
\end{verbatim}

\textit{WX, WY} \hspace{1cm} Input. REAL(8). Window coordinates for fill starting point.
BCOLOR

Input. INTEGER (4). RGB value of the boundary color.

Description
FLOODFILLRGB_W begins filling at the window-coordinate point (WX, WY). The fill color used by FLOODFILLRGB_W is set by SETCOLORRGB. You can obtain the current fill color by calling GETCOLORRGB.

If the starting point lies inside a figure, the interior is filled; if it lies outside a figure, the background is filled. In both cases, the fill color is the current color set by SETCOLORRGB. The starting point must be inside or outside the figure, not on the figure boundary itself. Filling occurs in all directions, stopping at pixels of the boundary color, BCOLOR.

Output
The result type is INTEGER (4). The result is a nonzero value if successful; otherwise, 0 (occurs if the fill could not be completed, or if the starting point lies on a pixel with the boundary color color, or if the starting point lies outside the clipping region). bounds, the return is 0.

GETARCINFO

Determines the endpoints (in viewport coordinates) of the most recently drawn arc or pie.

Prototype

INTERFACE
FUNCTION GETARCINFO (LPSTART, LPEND, LPPAINT)
INTEGER (2) GETARCINFO
STRUCTURE /XYCOORD/
    INTEGER (2) XCOORD
    INTEGER (2) YCOORD
END STRUCTURE


GETARCINFO updates the /XYCOORD/LPSTART and /XYCOORD/LPEND to contain the endpoints (in viewport coordinates) of the arc drawn by the most recent call to the ARC or PIE functions.

The returned value in LPPAINT specifies a point from which a pie can be filled. You can use this to fill a pie in a color different from the border color. After a call to GETARCINFO, change colors using SETCOLORRGB. Use the new color, along with the coordinates in LPPAINT, as arguments for the FLOODFILLRGB function.

Output

The result type is INTEGER(2). The result is nonzero if successful. The result is zero if neither the ARC nor the PIE function has been successfully called since the last time CLEARSCREEN or SETWINDOWCONFIG was successfully called, or since a new viewport was selected.
**GETBKCOLOR**

*Gets the current background color index for both text and graphics output.*

**Prototype**

```fortran
INTERFACE
  FUNCTION GETBKCOLOR()
    INTEGER(4) GENBKCOLOR
  END FUNCTION
END INTERFACE
```

**Description**

GETBKCOLOR returns the current background color index for both text and graphics, as set with **SETBKCOLOR**. The color index of text over the background color is set with **SETTEXTCOLOR** and returned with **GETTEXTCOLOR**. The color index of graphics over the background color is set with **SETCOLOR** and returned with **GETCOLOR**. These non-RGB color functions use color indexes, not true color values, and limit the user to colors in the palette, at most 256. For access to all system colors, use **SETBKCOLORRGB**, **SETCOLORRGB**, and **SETTEXTCOLORRGB**.

Generally, INTEGER(4) color arguments refer to color values and INTEGER(2) color arguments refer to color indexes. The two exceptions are **GETBKCOLOR** and **SETBKCOLOR**. The default background index is 0, which is associated with black unless the user remaps the palette with **REMAPPALETTERGB**.

**Output**

The result type is INTEGER(4). The result is the current background color index.
GETCOLOR

*Gets the current graphics color index.*

**Prototype**

```fortran
INTERFACE
    FUNCTION GETCOLOR()
        INTEGER(2) GETCOLOR
    END FUNCTION
END INTERFACE
```

**Description**

GETCOLOR returns the current color index used for graphics over the background color as set with SETCOLOR. The background color index is set with SETBKCOLOR and returned with GETBKCOLOR. The color index of text over the background color is set with SETTEXTCOLOR and returned with GETTEXTCOLOR. These non-RGB color functions use color indexes, not true color values, and limit the user to colors in the palette, at most 256. For access to all system colors, use SETCOLORRGB, SETBKCOLORRGB, and SETTEXTCOLORRGB.

**Output**

The result type is `INTEGER(2)`. The result is the current color index, if successful; otherwise, -1.
**GETCURRENTPOSITION**

*Get the coordinates of the current graphics position.*

**Prototype**

```
INTERFACE
    SUBROUTINE GETCURRENTPOSITION(S)
        STRUCTURE /XYCOORD/
            INTEGER(2) XCOORD, YCOORD
        END STRUCTURE
        RECORD /XYCOORD/ S
    END SUBROUTINE
END INTERFACE
```

**Output.** RECORD /XYCOORD/. Viewport coordinates of current graphics position.

**Description**

LINETO, MOVETO, and OUTGTEXT all change the current graphics position. It is in the center of the screen when a window is created.

Graphics output starts at the current graphics position returned by GETCURRENTPOSITION. This position is not related to normal text output (from OUTTEXT or WRITE, for example), which begins at the current text position (see SETTEXTPOSITION). It does, however, affect graphics text output from OUTGTEXT.
GETCURRENTPOSITION_W

Get the coordinates of the current graphics position.

Prototype

INTERFACE
    SUBROUTINE GETCURRENTPOSITION_W(S)
        STRUCTURE /WXYCOORD/
            DOUBLE PRECISION WX, YX
        END STRUCTURE
        RECORD /WXYCOORD/ S
    END SUBROUTINE
END INTERFACE

S        Output. RECORD /WXYCOORD/. Window coordinates of current graphics position.

Description

LINETO, MOVETO, and OUTGTEXT all change the current graphics position. It is in the center of the screen when a window is created.

Graphics output starts at the current graphics position returned by GETCURRENTPOSITION_W. This position is not related to normal text output (from OUTTEXT or WRITE, for example), which begins at the current text position (see SETTEXTPosition). It does, however, affect graphics text output from OUTGTEXT.
GETFILLMASK

*Returns the current pattern used to fill shapes.*

**Prototype**

```fortran
INTERFACE
    SUBROUTINE GETFILL(MASK)
        INTEGER(1) MASK(8)
    END SUBROUTINE
END INTERFACE
```

**Mask**

*Output. INTEGER(1). One-dimensional array of length 8.*

**Description**

There are 8 bytes in MASK, and each of the 8 bits in each byte represents a pixel, creating an 8x8 pattern. The first element (byte) of MASK becomes the top 8 bits of the pattern, and the eighth element (byte) of MASK becomes the bottom 8 bits.

During a fill operation, pixels with a bit value of 1 are set to the current graphics color, while pixels with a bit value of 0 are unchanged. The current graphics color is set with `SETCOLORRGB` or `SETCOLOR`. The 8-byte mask is replicated over the entire fill area. If no fill mask is set (with `SETFILLMASK`), or if the mask is all ones, solid current color is used in fill operations.

The fill mask controls the fill pattern for graphics routines (`FLOODFILLRGB`, `PIE`, `ELLIPSE`, `POLYGON`, and `RECTANGLE`).
GETIMAGE

Stores the screen image defined by a specified bounding rectangle.

Prototype

INTERFACE
SUBROUTINE GETIMAGE(X1,Y1,X2,Y2,IMAGE)
INTEGER(2) X1,Y1,X2,Y2
INTEGER(1) IMAGE(*)
END SUBROUTINE
END INTERFACE

X1, Y1  Input. INTEGER(2). Viewport coordinates for upper-left corner of bounding rectangle.
X2, Y2  Input. INTEGER(2). Viewport coordinates for lower-right corner of bounding rectangle.

Description

GETIMAGE defines the bounding rectangle in viewport-coordinate points (X1, Y1) and (X2, Y2).

The buffer used to store the image must be large enough to hold it. You can determine the image size by calling IMAGESIZE at run time, or by using the formula described under IMAGESIZE. After you have determined the image size, you can dimension the buffer accordingly.
GETIMAGE_W

Stores the screen image defined by a specified bounding rectangle.

Prototype

INTERFACE
SUBROUTINE GETIMAGE_W(WX1,WY1,WX2,WY2,IMAGE)
double precision WX1,WY1,WX2,WY2
INTEGER(1) IMAGE(*)
!MS$ ATTRIBUTES REFERENCE :: IMAGE
END SUBROUTINE
END INTERFACE

WX1, WY1    Input. REAL(8). Window coordinates for upper-left corner of bounding rectangle.
WX2, WY2    Input. REAL(8). Window coordinates for lower-right corner of bounding rectangle.

Description

GETIMAGE_W defines the bounding rectangle in window-coordinate points (WX1, WY1) and (WX2, WY2).

The buffer used to store the image must be large enough to hold it. You can determine the image size by calling IMAGESIZE at run time, or by using the formula described under IMAGESIZE. After you have determined the image size, you can dimension the buffer accordingly.
**GETLINESTYLE**

*Returns the current graphics line style.*

**Prototype**

```
INTERFACE
  FUNCTION GETLINESTYLE()
    INTEGER(2) GETLINESTYLE
  END FUNCTION
END INTERFACE
```

**Description**

GETLINESTYLE retrieves the mask (line style) used for line drawing. The mask is a 16-bit number, where each bit represents a pixel in the line being drawn.

If a bit is 1, the corresponding pixel is colored according to the current graphics color and logical write mode; if a bit is 0, the corresponding pixel is left unchanged. The mask is repeated for the entire length of the line. The default mask is `#FFFF` (a solid line). A dashed line can be represented by `#FF00` (long dashes) or `#F0F0` (short dashes).

The line style is set with SETLINESTYLE. The current graphics color is set with SETCOLORRGB or SETCOLOR. SETWRITEMODE affects how the line is displayed.

The line style retrieved by GETLINESTYLE affects the drawing of straight lines as in LINETO, POLYGON and RECTANGLE, but not the drawing of curved lines as in ARC, ELLIPSE or PIE.

**Output**

The result type is `INTEGER(2)`. The result is the current line style.
GETPHYSCOORD

Translates viewport coordinates to physical coordinates.

Prototype

INTERFACE
SUBROUTINE GETPHYSCOORD(X, Y, S)
   INTEGER(2) X, Y
   STRUCTURE /XYCOORD/
      INTEGER(2) XCOORD, YCOORD
   END STRUCTURE
   RECORD /XYCOORD/ S
END SUBROUTINE
END INTERFACE

X, Y Input. INTEGER(2). Viewport coordinates to be translated to physical coordinates.

S Output. RECORD /XYCOORD/. Physical coordinates of the input viewport position.

Description

Physical coordinates refer to the physical screen. Viewport coordinates refer to an area of the screen defined as the viewport with SETVIEWPORT. Both take integer coordinate values. Window coordinates refer to a window sized with SETWINDOW or SETWSIZEQQ. Window coordinates are floating-point values and allow easy scaling of data to the window area.
GETPIXEL

Returns the color index of the pixel at a specified location.

Prototype

INTERFACE
  FUNCTION GETPIXEL(X, Y)
    INTEGER(2) GETPIXEL, X, Y
  END FUNCTION
END INTERFACE

X, Y       Input. INTEGER(2). Viewport coordinates for pixel position.

Description

Color routines without the RGB suffix, such as GETPIXEL, use color indexes, not true color values, and limit you to colors in the palette, at most 256. To access all system colors, use SETPIXELRGB to specify an explicit Red-Green-Blue value and retrieve the value with GETPIXELRGB.

Output

The result type is INTEGER(2). The result is the pixel color index if successful; otherwise, -1 (if the pixel lies outside the clipping region, for example).
GETPIXEL_W

Returns the color index of the pixel at a specified location.

Prototype

```
INTERFACE
  FUNCTION GETPIXEL_W(WX,WY)
    INTEGER(2) GETPIXEL_W
    DOUBLE PRECISION WX,WY
  END FUNCTION
END INTERFACE
```

**Input**

- **REAL (8)**. Window coordinates for pixel position.

**Description**

Color routines without the RGB suffix, such as GETPIXEL_W, use color indexes, not true color values, and limit you to colors in the palette, at most 256. To access all system colors, use SETPIXELRGB to specify an explicit Red-Green-Blue value and retrieve the value with GETPIXELRGB.

**Output**

The result type is **INTEGER (2)**. The result is the pixel color index if successful; otherwise, -1 (if the pixel lies outside the clipping region, for example).
GETPIXELS

*Gets the color indexes of multiple pixels.*

**Prototype**

```fortran
INTERFACE
SUBROUTINE GETPIXELS(N, X, Y, C)
  INTEGER(4) N ! input : size of arrays
  INTEGER(2) X(*) ! input : x coordinates
  INTEGER(2) Y(*) ! input : y coordinates
  INTEGER(2) C(*) ! input : palette indices
END SUBROUTINE
END INTERFACE
```

**N**

Input. INTEGER (4). Number of pixels to get. Sets the number of elements in the other arguments.

**X, Y**

Input. INTEGER (2). Parallel arrays containing viewport coordinates of pixels to get.

**C**

Output. INTEGER (2). Array to be filled with the color indexes of the pixels at *x* and *y*.

**Description**

GETPIXELS fills in the array COLOR with color indexes of the pixels specified by the two input arrays *X* and *Y*. These arrays are parallel: the first element in each of the three arrays refers to a single pixel, the second element refers to the next pixel, and so on.

If the pixel is outside the clipping region, the value placed in the *color* array is undefined. Calls to GETPIXELS with *N* less than 1 are ignored.

GETPIXELS is a much faster way to acquire multiple pixel color indexes than individual calls to GETPIXEL.
The range of possible pixel color index values is determined by the current video mode and palette, at most 256 colors. To access all system colors you need to specify an explicit Red-Green-Blue (RGB) value with an RGB color function such as SETPIXELSRGB and retrieve the value with GETPIXELSRGB, rather than a palette index with a non-RGB color function.

---

**GETTEXTCOLOR**

*Gets the current text color index.*

**Prototype**

```fortran
INTERFACE
    FUNCTION GETTEXTCOLOR()
        INTEGER(2) GETTEXTCOLOR
    END FUNCTION
END INTERFACE
```

**Description**

GETTEXTCOLOR returns the text color index set by SETTEXTCOLOR. SETTEXTCOLOR affects text output with OUTTEXT, WRITE, and PRINT. The background color index is set with SETBKCOLOR and returned with GETBKCOLOR. The color index of graphics over the background color is set with SETCOLOR and returned with GETCOLOR. These non-RGB color functions use color indexes, not true color values, and limit the user to colors in the palette, at most 256. To access all system colors, use SETTEXTCOLORRGB, SETBKCOLORRGB, and SETCOLORRGB.

The default text color index is 15, which is associated with white unless the user remaps the palette.

**Output**

The result type is INTEGER(2). It is the current text color index.
GETTEXTPOSITION

*Returns the current text position.*

**Prototype**

```fortran
INTERFACE
SUBROUTINE GETTEXTPOSITION(S)
  STRUCTURE /RCCOORD/
    INTEGER(2) ROW, COL
  END STRUCTURE
  RECORD /RCCOORD/ S
END SUBROUTINE
END INTERFACE
```

**S**

Output. RECORD /RCCOORD/. Current text position.

**Description**

The text position given by coordinates (1, 1) is defined as the upper-left corner of the text window. Text output from the OUTTEXT function (and WRITE and PRINT statements) begins at the current text position. Font text is not affected by the current text position. Graphics output, including OUTGTEXT output, begins at the current graphics output position, which is a separate position returned by GETCURRENTPOSITION.

GETTEXTWINDOW

*Finds the boundaries of the current text window.*

**Prototype**

```fortran
INTERFACE
```

---

4-40
SUBROUTINE gettextwindow(R1,C1,R2,C2)
INTEGER(2) R1,C1,R2,C2
END SUBROUTINE
END INTERFACE

\textbf{R1, C1} \quad \textbf{Output. INTEGER(2). Row and column coordinates for upper-left corner of the text window.}

\textbf{R2, C2} \quad \textbf{Output. INTEGER(2). Row and column coordinates for lower-right corner of the text window.}

\textbf{Description}

Output from \texttt{OUTTEXT} and \texttt{WRITE} is limited to the text window. By default, this is the entire window, unless the text window is redefined by \texttt{SETTEXTWINDOW}.

The window defined by \texttt{SETTEXTWINDOW} has no effect on output from \texttt{OUTGTEXT}.

---

\textbf{GETVIEWCOORD}

\textit{Translates physical coordinates or window coordinates to viewport coordinates.}

\textbf{Prototype}

\begin{verbatim}
INTERFACE GETVIEWCOORD
SUBROUTINE GETVIEWCOORD(X,Y,S)
INTEGER(2) X, Y
structure /XYCOORD/
   INTEGER(2) XCOORD, YCOORD
END STRUCTURE
record /XYCOORD/ S
END SUBROUTINE
\end{verbatim}
Description
Viewport coordinates refer to an area of the screen defined as the viewport with `SETVIEWPORT`. Physical coordinates refer to the whole screen. Both take integer coordinate values. Window coordinates refer to a window sized with `SETWINDOW` or `SETWSIZEQQ`. Window coordinates are floating-point values and allow easy scaling of data to the window area.

**GETVIEWCOORD_W**

*Translates physical coordinates or window coordinates to viewport coordinates.*

**Prototype**

```fortran
END INTERFACE
SX, SY Input. INTEGER(2). Physical coordinates to be converted to viewport coordinates.
S Output. RECORD /XYCOORD/. Viewport coordinates.

Description
Viewport coordinates refer to an area of the screen defined as the viewport with `SETVIEWPORT`. Physical coordinates refer to the whole screen. Both take integer coordinate values. Window coordinates refer to a window sized with `SETWINDOW` or `SETWSIZEQQ`. Window coordinates are floating-point values and allow easy scaling of data to the window area.

**GETVIEWCOORD_W**

*Translates physical coordinates or window coordinates to viewport coordinates.*

**Prototype**

```fortran
END INTERFACE
WX, WY Input. REAL(8). Window coordinates to be converted to viewport coordinates.
S Output. RECORD /WXCOORD/. Viewport coordinates.
Description
Viewport coordinates refer to an area of the screen defined as the viewport with \texttt{SETVIEWPORT}. Physical coordinates refer to the whole screen. Both take integer coordinate values. Window coordinates refer to a window sized with \texttt{SETWINDOW} or \texttt{SETWSIZEQQ}. Window coordinates are floating-point values and allow easy scaling of data to the window area.

GETWINDOWCOORD
\textit{Translates viewport coordinates to window coordinates.}

Prototype
\begin{verbatim}
INTERFACE GETWINDOWCOORD
  SUBROUTINE GETWINDOWCOORD(X,Y,S)
    REAL(4) X, Y
    STRUCTURE /XYCOORD/ 
      REAL(4) X, Y
    END STRUCTURE
    RECORD /XYCOORD/ S
  END SUBROUTINE
END INTERFACE
\end{verbatim}

\begin{itemize}
  \item \texttt{X, Y} \hspace{1cm} \texttt{Input. REAL(4). Viewport coordinates to be converted to window coordinates.}
  \item \texttt{S} \hspace{1cm} \texttt{Output. RECORD /XYCOORD/. Window coordinates.}
\end{itemize}

Description
Physical coordinates refer to the physical screen. Viewport coordinates refer to an area of the screen defined as the viewport with \texttt{SETVIEWPORT}. Both take integer coordinate values. Window coordinates refer to a window sized with \texttt{SETWINDOW} or \texttt{SETWSIZEQQ}. Window coordinates are floating-point values and allow easy scaling of data to the window area.
GETWRITEMODE

*Returns the current logical write mode.*

**Prototype**

```fortran
INTERFACE
    FUNCTION GETWRITEMODE()
        INTEGER(2) GETWRITEMODE
    END FUNCTION
END INTERFACE
```

**Description**

Returns the current logical write mode, which is used when drawing lines with the LINETO, POLYGON, and RECTANGLE functions. The write mode is set with SETWRITEMODE.

**Output**

The result type is `INTEGER(2)`. The result is the current write mode. The default value is `$GPSET$. Possible return values are:

- `$GPSET` Causes lines to be drawn in the current graphics color. (default)
- `$GAND` Causes lines to be drawn in the color that is the logical AND of the current graphics color and the current background color.
- `$GOR` Causes lines to be drawn in the color that is the logical OR of the current graphics color and the current background color.
- `$GPRESET` Causes lines to be drawn in the color that is the logical NOT of the current graphics color.
- `$GXOR` Causes lines to be drawn in the color that is the logical exclusive OR (XOR) of the current graphics color and the current background color.
GRSTATUS

Returns the status of the most recently used graphics routine.

Prototype

INTERFACE

FUNCTION GRSTATUS()
  integer*2 GRSTATUS
END FUNCTION
END INTERFACE

Description

Use GRSTATUS immediately following a call to a graphics routine to determine if errors or warnings were generated. Return values less than 0 are errors, and values greater than 0 are warnings.

$GRFILEWRITEERROR  Error writing bitmap file
$GRFILEOPENERROR   Error opening bitmap file
$GRIMAGEREADERERROR Error reading image
$GRBITMAPDISPLAYERROR Error displaying bitmap
$GRBITMAPTOOLARGE   Bitmap too large
$GRIMPROPERBITMAPFORMAT  Improper format for bitmap file
$GRFILEREADERROR    Error reading file
$GRNOBITMAPFILE     No bitmap file
$GRINVALIDIMAGEBUFFER  Image buffer data inconsistent
$GRINSUFFICIENTMEMORY  Not enough memory to allocate buffer or to complete a fill operation
$GRINVALIDPARAMETER  One or more parameters invalid
$GRMODENOTSUPPORTED  Requested video mode not supported
$GRERROR            Graphics error
$GROK\quad$Success
$GRNOOUTPUT\quad$No action taken
$GRCLIPPED\quad$Output was clipped to viewport
$GRPARAMETERALTERED\quad$One or more input parameters was altered to be within range, or pairs of parameters were interchanged to be in the proper order

After a graphics call, compare the return value of $GRSTATUS$ to $GROK$ to determine if an error has occurred.

**Output**
The result type is $\text{INTEGER(2)}$. The result is the status of the most recently used graphics function.

### IMAGESIZE

*Returns the number of bytes needed to store the image inside the specified bounding rectangle.*

**Prototype**

```fortran
INTERFACE
    FUNCTION IMAGESIZE(X1,Y1,X2,Y2)
        INTEGER(4) IMAGESIZE
        INTEGER(2) X1,Y1,X2,Y2
    END FUNCTION
END INTERFACE
```

**Input**

- $X1, Y1$: INTEGER(2). Viewport coordinates for upper-left corner of image.
- $X2, Y2$: Integer(2). Viewport coordinates for lower-right corner of image.
**Description**

IMAGESIZE defines the bounding rectangle in viewport-coordinate points \((X1, Y1)\) and \((X2, Y2)\). Returns the number of bytes needed to store the image inside the specified bounding rectangle. Useful for determining how much memory is needed for a call to GETIMAGE.

**Output**

The result type is `INTEGER(4)`. The result is the storage size of an image in bytes.

---

**IMAGESIZE_W**

*Returns the number of bytes needed to store the image inside the specified bounding rectangle.*

---

**Prototype**

```fortran
INTERFACE
  FUNCTION IMAGESIZE_W(WX1,WY1,WX2,WY2)
    INTEGER(4) IMAGESIZE_W
    DOUBLE PRECISION WX1,WY1,WX2,WY2
  END FUNCTION
END INTERFACE
```

**Input**

- `REAL(8)` `WX1, WY1`: Window coordinates for upper-left corner of image.
- `REAL(8)` `WX2, WY2`: Window coordinates for lower-right corner of image.

**Description**

IMAGESIZE_W defines the bounding rectangle in terms of window-coordinate points \((WX1, WY1)\) and \((WX2, WY2)\).
The function returns the number of bytes needed to store the image inside the specified bounding rectangle. IMAGESIZE is useful for determining how much memory is needed for a call to GETIMAGE.

**Output**

The result type is INTEGER(4). The result is the storage size of an image in bytes.

**LINETO**

*Draws a line from the current graphics position up to and including the end point.*

**Prototype**

```
INTERFACE
  FUNCTION LINETO(X,Y)
    INTEGER(2) LINETO,X,Y
  END FUNCTION
END INTERFACE
```

**X, Y**

Input. INTEGER(2). Viewport coordinates of end point.

**Description**

The line is drawn using the current graphics color, logical write mode, and line style. The graphics color is set with SETCOLORRGB, the write mode with SETWRITEMODE, and the line style with SETLINESTYLE.

If no error occurs, LINETO sets the current graphics position to the viewport point (X, Y).

If you use FLOODFILLRGB to fill in a closed figure drawn with LINETO, the figure must be drawn with a solid line style. Line style is solid by default and can be changed with SETLINESTYLE.
Output
The result type is INTEGER(2). The result is a nonzero value if successful; otherwise, 0.

LINETO_W
Draws a line from the current graphics position up to and including the end point.

Prototype

```fortran
INTERFACE
  FUNCTION LINETO_W(WX, WY)
    INTEGER(2) LINETO_W
    DOUBLE PRECISION WX, WY
  END FUNCTION
END INTERFACE
```

WX, WY
Input. REAL(8). Window coordinates of end point.

Description
The line is drawn using the current graphics color, logical write mode, and line style. The graphics color is set with SETCOLORRGB, the write mode with SETWRITEMODE, and the line style with SETLINESTYLE.
If no error occurs, LINETO_W sets the current graphics position to the window point (WX, WY).
If you use FLOODFILLRGB to fill in a closed figure drawn with LINETO_W, the figure must be drawn with a solid line style. Line style is solid by default and can be changed with SETLINESTYLE.

Output
The result type is INTEGER(2). The result is a nonzero value if successful; otherwise, 0.
LINETOAR

Draws a line between points in one array and corresponding points in another array.

Prototype

```
INTERFACE
  FUNCTION LINETOAR (X1,Y1,X2,Y2,CNT)  
    INTEGER(2) LINETOAR X1,Y1,X2,Y2  
    INTEGER(4) CNT
  END FUNCTION
END INTERFACE
```

X1, Y1
Input. INTEGER(2). From Viewport coordinates array.

X2, Y2
Input. INTEGER(2). To Viewport coordinates array.

CNT
Input. INTEGER(4). Length of each coordinate array; all should be the same size.

Description

Draws a line between each x, y point in the from-array to each corresponding x, y point in the to-array. The lines are drawn using the current graphics color, logical write mode, and line style. The graphics color is set with SETCOLORRGB, the write mode with SETWRITEMODE, and the line style with SETLINESTYLE.

Output

The result type is INTEGER(2). The result is a nonzero value if successful; otherwise, zero.
**LINETOAREX**

*Draws a line between points in one array and corresponding points in another array with specified line color and style.*

**Prototype**

```plaintext
INTERFACE
  FUNCTION LINETOAR (X1,Y1,X2,Y2,C,S,CNT)
    INTEGER(2) LINETOAR X1,Y1,X2,Y2
    INTEGER(4) C, S, CNT
  END FUNCTION
END INTERFACE
```

- **X1, Y1** Input. INTEGER(2). From Viewport coordinates array.
- **X2, Y2** Input. INTEGER(2). To Viewport coordinates array.
- **C** Input. INTEGER(4). Color array.
- **S** Input. INTEGER(4). Style array.
- **CNT** Input. INTEGER(4). Length of each coordinate array; all should be the same size.

**Description**

Draws a line between each x, y point in the from-array to each corresponding x, y point in the to-array. Each line is drawn with the specified graphics color and line style. The lines are drawn using the specified graphics colors and line styles, and with the current write mode. The current write mode is set with `SETWRITEMODE`.

**Output**

The result type is INTEGER(2). The result is a nonzero value if successful; otherwise, zero.
LOADIMAGE

Reads an image from a Windows bitmap file and displays it at a specified location.

Prototype

INTERFACE
  FUNCTION LOADIMAGE(FNAME, X, Y)
    INTEGER(4) LOADIMAGE, X, Y
    CHARACTER(LEN=*) FNAME
  END FUNCTION
END INTERFACE

FNAME  Input. CHARACTER(LEN=*) Path of the bitmap file.
X, Y   Input. INTEGER(4). Viewport coordinates for upper-left corner of image display.

Description

The image is displayed with the colors in the bitmap file. If the color palette in the bitmap file is different from the current system palette, the current palette is discarded and the bitmap's palette is loaded.

LOADIMAGE specifies the screen placement of the image in viewport coordinates.

Output

The result type is INTEGER(4). The result is zero if successful; otherwise, a negative value.
LOADIMAGE_W

*Reads an image from a Windows bitmap file and displays it at a specified location.*

**Prototype**

```
INTERFACE
  FUNCTION LOADIMAGE_W(FNAME, WX, WY)
    INTEGER(4) LOADIMAGE_W
    CHARACTER(LEN=*) FNAME
    DOUBLE PRECISION WX, WY
  END FUNCTION
END INTERFACE
```

FNAME  Input. CHARACTER(LEN=*). Path of the bitmap file.
WX , WY Input. REAL(8). Window coordinates for upper-left corner of image display.

**Description**

The image is displayed with the colors in the bitmap file. If the color palette in the bitmap file is different from the current system palette, the current palette is discarded and the bitmap's palette is loaded.

LOADIMAGE_W specifies the screen placement of the image in window coordinates.

**Output**

The result type is INTEGER (4). The result is zero if successful; otherwise, a negative value.
MOVETO

Moves the current graphics position to a specified point. No drawing occurs.

Prototype

```
INTERFACE MOVETO
  SUBROUTINE MOVETO(X, Y, S)
    INTEGER(2) X, Y
    integer*2 y
    STRUCTURE /XYCOORD/
      INTEGER(2) XCOORD, YCOORD
    END STRUCTURE
    RECORD/XYCOORD/S
  END SUBROUTINE
END INTERFACE
```

X, Y  Input. INTEGER(2). Viewport coordinates of the new graphics position.
S    Output. RECORD/XYCOORD/. Viewport coordinates of the previous graphics position.

Description

MOVETO sets the current graphics position to the viewport coordinate \((X, Y)\). It assigns the coordinates of the previous position to \(S\) respectively.
**MOVETO_W**

*Moves the current graphics position to a specified point. No drawing occurs.*

**Prototype**

```plaintext
INTERFACE
SUBROUTINE MOVETO_W(WX,WY,S)
  DOUBLE PRECISION WX,WY
  STRUCTURE /WXYCOORD/
    DOUBLE PRECISION WX, WY
  END STRUCTURE
  RECORD /WXYCOORD/ s
END SUBROUTINE
END INTERFACE
```

**Input.** REAL(8). Window coordinates of the new graphics position.

**Output.** RECORD /WXYCOORD/. Window coordinates of the previous graphics position.

**Description**

MOVETO_W sets the current graphics position to the window coordinates (WX, WY). Next call to MOVETO_W assigns the coordinates of the previous position to S, respectively.
OUTTEXT

In text or graphics mode, sends a string of text to the screen, including any trailing blanks.

Prototype

```fortran
INTERFACE
  SUBROUTINE OUTTEXT (TEXT)
    CHARACTER(LEN=*) TEXT
  END SUBROUTINE
END INTERFACE
```

TEXT Input. CHARACTER (LEN=*). String to be displayed.

Description

Text output begins at the current text position in the color set with SETTEXTCOLORRGB or SETTEXTCOLOR. No formatting is provided. After it outputs the text, OUTTEXT updates the current text position.

PIE

Draws a pie-shaped wedge in the current graphics color.

Prototype

```fortran
INTERFACE
  FUNCTION PIE(I,X1,Y1,X2,Y2,X3,Y3,X4,Y4)
    INTEGER(2) PIE, I, X1, Y1, X2, Y2, X3, Y3, X4, Y4
  END FUNCTION
END INTERFACE
```
I  Input. INTEGER(2). Fill flag. One of the following symbolic constants:
   $GFILLINTERIOR   Fills the figure using the current color and fill mask.
   $GBORDER         Does not fill the figure.

X1,Y1 Input. INTEGER(2). Viewport coordinates for upper-left corner of bounding rectangle.

X2,Y2 Input. INTEGER(2). Viewport coordinates for lower-right corner of bounding rectangle.

X3,Y3 Input. INTEGER(2). Viewport coordinates of start vector.

X4,Y4 Input. INTEGER(2). Viewport coordinates of end vector.

Description
The border of the pie wedge is drawn in the current color set by SETCOLORRGB.

The PIE function uses the viewport-coordinate system. The center of the arc is the center of the bounding rectangle, which is specified by the viewport-coordinate points (X1, Y1) and (X2, Y2). The arc starts where it intersects an imaginary line extending from the center of the arc through (X3, Y3). It is drawn counterclockwise about the center of the arc, ending where it intersects an imaginary line extending from the center of the arc through (X4, Y4).

The fill flag option $GFILLINTERIOR is equivalent to a subsequent call to FLOODFILLRGB using the center of the pie as the starting point and the current graphics color (set by SETCOLORRGB) as the fill color. If you want a fill color different from the boundary color, you cannot use the $GFILLINTERIOR option. Instead, after you have drawn the pie wedge, change the current color with SETCOLORRGB and then call FLOODFILLRGB. You must supply FLOODFILLRGB with an interior point in the figure you want to fill. You can get this point for the last drawn pie or arc by calling GETARCINFO.
If you fill the pie with \texttt{FLOODFILLRGB}, the pie must be bordered by a solid line style. Line style is solid by default and can be changed with \texttt{SETLINESTYLE}.

**Output**

The result type is \texttt{INTEGER(2)}. The result is nonzero if successful; otherwise, 0. If the pie is clipped or partially out of bounds, the pie is considered successfully drawn and the return is 1. If the pie is drawn completely out of bounds, the return is 0.

**PIE\_W**

\textit{Draws a pie-shaped wedge in the current graphics color.}

**Prototype**

\begin{verbatim}
INTERFACE
  FUNCTION PIE_W(I,WX1,WY1,WX2,WY2,WX3,WY3,WX4,WY4)
    INTEGER(2) PIE_W, I
    DOUBLE PRECISION WX1,WY1,WX2,WY2,WX3,WY3,WX4,WY4
  END FUNCTION
END INTERFACE
\end{verbatim}

\textbf{I} \hspace{1cm} Input. \texttt{INTEGER(2)}. Fill flag. One of the following symbolic constants:

- \texttt{$GFILLINTERIOR$} Fills the figure using the current color and fill mask.
- \texttt{$GBORDER$} Does not fill the figure.

\textbf{WX1,WY1} \hspace{1cm} Input. \texttt{REAL(8)}. Window coordinates for upper-left corner of bounding rectangle.

\textbf{WX2,WY2} \hspace{1cm} Input. \texttt{REAL(8)}. Window coordinates for lower-right corner of bounding rectangle.
**Description**

The border of the pie wedge is drawn in the current color set by `SETCOLORRGB`.

The `PIE_W` function uses the window-coordinate system. The center of the arc is the center of the bounding rectangle specified by the window-coordinate points \((WX1, WY1)\) and \((WX2, WY2)\). The arc starts where it intersects an imaginary line extending from the center of the arc through \((WX3, WY3)\). It is drawn counterclockwise about the center of the arc, ending where it intersects an imaginary line extending from the center of the arc through \((WX4, WY4)\).

The fill flag option `$GFILLINTERIOR` is equivalent to a subsequent call to `FLOODFILLRGB` using the center of the pie as the starting point and the current graphics color (set by `SETCOLORRGB`) as the fill color. If you want a fill color different from the boundary color, you cannot use the `$GFILLINTERIOR` option. Instead, after you have drawn the pie wedge, change the current color with `SETCOLORRGB` and then call `FLOODFILLRGB`. You must supply `FLOODFILLRGB` with an interior point in the figure you want to fill. You can get this point for the last drawn pie or arc by calling `GETARCINFO`.

If you fill the pie with `FLOODFILLRGB`, the pie must be bordered by a solid line style. Line style is solid by default and can be changed with `SETLINESTYLE`.

**Output**

The result type is `INTEGER(2)`. The result is nonzero if successful; otherwise, 0. If the pie is clipped or partially out of bounds, the pie is considered successfully drawn and the return is 1. If the pie is drawn completely out of bounds, the return is 0.
POLYGON

Draws a polygon using the current graphics color, logical write mode, and line style.

Prototype

INTERFACE
  FUNCTION POLYGON(CONTROL, LPPOINTS, CPOINTS)
  INTEGER(2) POLYGON, CONTROL, CPOINTS
  STRUCTURE /XYCOORD/ 
    INTEGER(2) XCOORD
    INTEGER(2) YCOORD
  END STRUCTURE
  RECORD /XYCOORD/ LPPOINTS(*)
  END FUNCTION
END INTERFACE

CONTROL      Input. INTEGER(2). Fill flag. One of the following symbolic constants:
              $GFILLINTERIOR Fills the figure using the current color and fill mask.
              $GBORDER Does not fill the figure.

LPPOINTS     Input. RECORD /XYCOORD/. Array defining the polygon vertices in viewport coordinates.

CPOINTS      Input. INTEGER(2). Number of polygon vertices.

Description

The border of the polygon is drawn in the current graphics color, logical write mode, and line style, set with \texttt{SETCOLORRGB}, \texttt{SETWRITEMODE}, and \texttt{SETLINESTYLE}, respectively. The \texttt{POLYGON} routine uses the viewport-coordinate system (expressed as \texttt{RECORD/XYCOORD/}).
The arguments LPPOINTS are arrays whose elements are RECORD /XYCOORD/ or RECORD /WXYCOORD/. Each element specifies one of the polygon's vertices. The argument CPOINTS is the number of elements (the number of vertices) in the LPPOINTS array.

Note that POLYGON draws between the vertices in their order in the array. Therefore, when drawing outlines, skeletal figures, or any other figure that is not filled, you need to be careful about the order of the vertices. If you don't want lines between some vertices, you may need to repeat vertices to make the drawing backtrack and go to another vertex to avoid drawing across your figure. Also, POLYGON draws a line from the last specified vertex back to the first vertex.

If you fill the polygon using FLOODFILLRGB, the polygon must be bordered by a solid line style. Line style is solid by default and can be changed with SETLINESTYLE.

Output
The result type is INTEGER(2). The result is nonzero if anything is drawn; otherwise, 0.

**POLYGON_W**

*Draws a polygon using the current graphics color, logical write mode, and line style.*

**Prototype**

```fortran
INTERFACE
FUNCTION POLYGON_W(CONTROL, LPPOINTS, CPOINTS)
  INTEGR(2) POLYGON_W, CONTROL, CPOINTS
  STRUCTURE /WXYCOORD/
    DOUBLE PRECISION WX, WY
END STRUCTURE
```
RECORD /WXYCOORD/LPPOINTS(*)
END FUNCTION
END INTERFACE

CONTROL
   Input. INTEGER(2). Fill flag. One of the following symbolic constants:
   $GFILLINTERIOR Fills the figure using the current color and fill mask.
   $GBORDER Does not fill the figure.

LPPOINTS
   Input. RECORD /WXYCOORD/. Array defining the polygon vertices in window coordinates.

CPOINTS
   Input. INTEGER(2). Number of polygon vertices.

Description
The border of the polygon is drawn in the current graphics color, logical write mode, and line style, set with SETCOLORRGB, SETWRITEMODE, and SETLINESTYLE, respectively. The POLYGON_W routine uses real-valued window coordinates (expressed as RECORD/WXYCOORD/).

The arguments LPPOINTS are arrays whose elements are RECORD /XYCOORD/ or RECORD /WXYCOORD/. Each element specifies one of the polygon’s vertices. The argument CPOINTS is the number of elements (the number of vertices) in the LPPOINTS array.

Note that POLYGON_W draws between the vertices in their order in the array. Therefore, when drawing outlines, skeletal figures, or any other figure that is not filled, you need to be careful about the order of the vertices. If you don't want lines between some vertices, you may need to repeat vertices to make the drawing backtrack and go to another vertex to avoid drawing across your figure. Also, POLYGON_W draws a line from the last specified vertex back to the first vertex.

If you fill the polygon using FLOODFILLRGB, the polygon must be bordered by a solid line style. Line style is solid by default and can be changed with SETLINESTYLE.
Output
The result type is INTEGER(2). The result is nonzero if anything is drawn; otherwise, 0.

PUTIMAGE
Transfers the image stored in memory to the screen.

Prototype

INTERFACE
  SUBROUTINE PUTIMAGE(X,Y,IMAGE,ACTION)
    INTEGER(2) X,Y,ACTION
    INTEGER(1) IMAGE(*)
  END SUBROUTINE
END INTERFACE

X,Y      Input. INTEGER(2). Viewport coordinates for upper-left corner of the image when placed on the screen.
ACTION   Input. INTEGER(2). Interaction of the stored image with the existing screen image. One of the following symbolic constants:
          $GAND Forms a new screen display as the logical AND of the stored image and the existing screen display. Points that have the same color in both the existing screen image and the stored image remain the same color, while points that have different colors are joined by a logical AND.
$GOR Superimposes the stored image onto the existing screen display. The resulting image is the logical OR of the image.

$GPRESET Transfers the data point-by-point onto the screen. Each point has the inverse of the color attribute it had when it was taken from the screen by GETIMAGE, producing a negative image.

$GPSET Transfers the data point-by-point onto the screen. Each point has the exact color attribute it had when it was taken from the screen by GETIMAGE.

$GXOR Causes points in the existing screen image to be inverted wherever a point exists in the stored image. This behavior is like that of a cursor. If you perform an exclusive OR of an image with the background twice, the background is restored unchanged. This allows you to move an object around without erasing the background. The $GXOR constant is a special mode often used for animation.

In addition, the following ternary raster operation constants can be used (described in the online documentation for the WIN32 API BitBlt):

$GSRCCOPY (same as $GPSET)
$GSRCPAINT (same as $GOR)
$GSRCAND (same as $GAND)
$GSRCINVERT (same as $GXOR)
$GSRCERASE
$GNOTSRCCOPY (same as $GPRESET)
$GNOTSRCERASE
$GMERGECOPY
Description

PUTIMAGE places the upper-left corner of the image at the viewport coordinates \((X, Y)\).

**PUTIMAGE_W**

Transfers the image stored in memory to the screen.

**Prototype**

```fortran
INTERFACE
  SUBROUTINE PUTIMAGE_W(WX, WY, IMAGE, ACTION)
    DOUBLE PRECISION WX, WY
    INTEGER(1) IMAGE(*)
  !MS$ ATTRIBUTES REFERENCE :: IMAGE
    INTEGER(2) ACTION
  END SUBROUTINE
END INTERFACE
```

**Wx, Wy**

Input, REAL(8). REAL(8). Window coordinates for upper-left corner of the image when placed on the screen.
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>Input. INTEGER(2). Interaction of the stored image with the existing screen image. One of the following symbolic constants:</td>
</tr>
</tbody>
</table>

- **$\text{GAND}$**: Forms a new screen display as the logical AND of the stored image and the existing screen display. Points that have the same color in both the existing screen image and the stored image remain the same color, while points that have different colors are joined by a logical AND.

- **$\text{GOR}$**: Superimposes the stored image onto the existing screen display. The resulting image is the logical OR of the image.

- **$\text{GRESET}$**: Transfers the data point-by-point onto the screen. Each point has the inverse of the color attribute it had when it was taken from the screen by GETIMAGE_W, producing a negative image.

- **$\text{GPSET}$**: Transfers the data point-by-point onto the screen. Each point has the exact color attribute it had when it was taken from the screen by GETIMAGE_W.

- **$\text{GXOR}$**: Causes points in the existing screen image to be inverted wherever a point exists in the stored image. This behavior is like that of a cursor. If you perform an exclusive OR of an image with the background twice, the background is restored unchanged. This allows you to move an object around without erasing the background. The $\text{GXOR}$ constant is a special mode often used for animation.
In addition, the following ternary raster operation constants can be used (described in the online documentation for the WIN32 API BitBlt):

- \$GSRCCOPY (same as \$GPSET)
- \$GSRCPAINT (same as \$GOR)
- \$GSRCAND (same as \$GAND)
- \$GSRCINVERT (same as \$GXOR)
- \$GSRCERASE
- \$GNOTSRCCOPY (same as \$PRESET)
- \$GNOTSRCERASE
- \$GMERGECOPY
- \$GMERGEPAINT
- \$GPATCOPY
- \$GPATPAINT
- \$GPATINVERT
- \$GDSTINVERT
- \$GBLACKNESS
- \$GWHITENESS

**Description**

`PUTIMAGE_W` places the upper-left corner of the image at the window coordinates \((WX, WY)\).
RECTANGLE

Draws a rectangle using the current graphics color, logical write mode, and line style.

Prototype

```fortran
INTERFACE
  FUNCTION RECTANGLE(CONTROL, X1, Y1, X2, Y2)
    INTEGER(2) RECTANGLE
    INTEGER(2) CONTROL, X1, Y1, X2, Y2
  END FUNCTION
END INTERFACE
```

CONTROL

Input. INTEGER(2). Fill flag. One of the following symbolic constants:

- $GFILLINTERIOR: Draws a solid figure using the current color and fill mask.
- $GBORDER: Draws the border of a rectangle using the current color and line style.

X1, Y1

Input. INTEGER(2). Viewport coordinates for upper-left corner of rectangle.

X2, Y2

Input. INTEGER(2). Viewport coordinates for lower-right corner of rectangle.

Description

The RECTANGLE function uses the viewport-coordinate system. The viewport coordinates (X1, Y1) and (X2, Y2) are the diagonally opposed corners of the rectangle.

SETCOLORRGB sets the current graphics color. SETFILLMASK sets the current fill mask. By default, filled graphic shapes are filled solid with the current color.
If you fill the rectangle using \texttt{FLOODFILLRGB}, the rectangle must be bordered by a solid line style. Line style is solid by default and can be changed with \texttt{SETLINESTYLE}.

**Output**

The result type is \texttt{INTEGER(2)}. The result is nonzero if successful; otherwise, 0.

---

**RECTANGLE\_W**

*Draws a rectangle using the current graphics color, logical write mode, and line style.*

**Prototype**

```fortran
INTERFACE
  FUNCTION RECTANGLE\_W(CONTROL,WX1,WY1,WX2,WY2)
    INTEGER(2) RECTANGLE\_W,CONTROL
    DOUBLE PRECISION WX1,WY1,WX2,WY2
  END FUNCTION
END INTERFACE
```

- **CONTROL**
  
  Input. \texttt{INTEGER(2)}. Fill flag. One of the following symbolic constants:
  - \$GFILLINTERIOR: Draws a solid figure using the current color and fill mask.
  - \$GBORDER: Draws the border of a rectangle using the current color and line style.

- **WX1,WY1**
  Input. \texttt{REAL(8)}. Window coordinates for upper-left corner of rectangle.

- **WX2,WY2**
  Input. \texttt{REAL(8)}. Window coordinates for lower-right corner of rectangle.
Description
The RECTANGLE_W function uses the window-coordinate system. The window coordinates \((WX1, WY1)\) and \((WX2, WY2)\) are the diagonally opposed corners of the rectangle.

SETCOLORRGB sets the current graphics color. SETFILLMASK sets the current fill mask. By default, filled graphic shapes are filled solid with the current color.

If you fill the rectangle using FLOODFILLRGB, the rectangle must be bordered by a solid line style. Line style is solid by default and can be changed with SETLINESTYLE.

Output
The result type is INTEGER(2). The result is nonzero if successful; otherwise, 0.

REMAPALLPALETTERGB
Remaps a set of Red-Green-Blue (RGB) color values to indexes recognized by the video hardware.

Prototype
```
INTERFACE
FUNCTION REMAPALLPALETTERGB(COLORS)
  INTEGER(2) REMAPALLPALETTERGB
  INTEGER(4) COLORS(*)
END FUNCTION
END INTERFACE
```
COLORS
    Input. INTEGER(4). Ordered array of RGB color values to be mapped in order to indexes. Must hold 0-255 elements.
Description

The REMAPALLPALETTERGB function remaps all of the available color indexes simultaneously (up to 236; 20 indexes are reserved by the operating system). The COLORS argument points to an array of RGB color values.

The default mapping between the first 16 indexes and color values is shown in the following table.

<table>
<thead>
<tr>
<th>Index</th>
<th>Color</th>
<th>Index</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$BLACK</td>
<td>8</td>
<td>$GRAY</td>
</tr>
<tr>
<td>1</td>
<td>$BLUE</td>
<td>9</td>
<td>$LIGHTBLUE</td>
</tr>
<tr>
<td>2</td>
<td>$GREEN</td>
<td>10</td>
<td>$LIGHTGREEN</td>
</tr>
<tr>
<td>3</td>
<td>$CYAN</td>
<td>11</td>
<td>$LIGHTCYAN</td>
</tr>
<tr>
<td>4</td>
<td>$RED</td>
<td>12</td>
<td>$LIGHTRED</td>
</tr>
<tr>
<td>5</td>
<td>$MAGENTA</td>
<td>13</td>
<td>$LIGHTMAGENTA</td>
</tr>
<tr>
<td>6</td>
<td>$BROWN</td>
<td>14</td>
<td>$YELLOW</td>
</tr>
<tr>
<td>7</td>
<td>$WHITE</td>
<td>15</td>
<td>$BRIGHTWHITE</td>
</tr>
</tbody>
</table>

The number of colors mapped can be fewer than 236 if the number of colors supported by the current video mode is fewer, but at most 236 colors can be mapped by REMAPALLPALETTERGB. Most Windows graphics drivers support a palette of 256K colors or more, of which only a few can be mapped into the 236 palette indexes at a time. To access and use all colors on the system, bypass the palette and use direct RGB color functions such as SETCOLORRGB and SETPIXELSRGB.

In each RGB color value, each of the three colors, red, green and blue, is represented by an eight-bit value (2 hex digits). In the values you specify with REMAPALLPALETTERGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 11111111 (hex FF) the maximum for each of the three components. For example, #008080 yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFFFF full-intensity for all three, resulting in bright white.
Output
The result type is INTEGER(4). REMAPALLPALETTERGB returns 0 if successful; otherwise, -1.

REMAPPALETTERGB
Remaps one color index to an RGB color value.

Prototype

INTERFACE
FUNCTION REMAPPALETTERGB(INDEX, COLOR)
  INTEGER(4) REMAPPALETTERGB, COLOR
  INTEGER(2) INDEX
END FUNCTION
END INTERFACE

COLOR Input. INTEGER(4). RGB color value to assign to a color index.
INDEX Input. INTEGER(2). Color index to be reassigned an RGB color.

Description
The REMAPPALETTERGB function remaps one of the available color indexes (up to 236; 20 indexes are reserved by the operating system). The COLOR argument is the RGB color value to assign. The default mapping between the first 16 indexes and color values is shown in the following table.

<table>
<thead>
<tr>
<th>Index</th>
<th>Color</th>
<th>Index</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$BLACK</td>
<td>8</td>
<td>$GRAY</td>
</tr>
<tr>
<td>1</td>
<td>$BLUE</td>
<td>9</td>
<td>$LIGHTBLUE</td>
</tr>
<tr>
<td>2</td>
<td>$GREEN</td>
<td>10</td>
<td>$LIGHTGREEN</td>
</tr>
</tbody>
</table>
The number of colors mapped can be fewer than 236 if the number of colors supported by the current video mode is fewer, but at most 236 colors can be mapped by REMAPPALETTERGB. Most Windows graphics drivers support a palette of 256K colors or more, of which only a few can be mapped into the 236 palette indexes at a time. To access and use all colors on the system, bypass the palette and use direct RGB color functions such as such as SETCOLORRGB and SETPIXELSRGB.

In each RGB color value, each of the three colors, red, green and blue, is represented by an eight-bit value (2 hex digits). In the values you specify with REMAPPALETTERGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 11111111 (hex FF) the maximum for each of the three components. For example, #008080 yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

**Output**

The result type is INTEGER (4). REMAPPALETTERGB returns the previous color assigned to the index.
SAVEIMAGE

Saves an image from a specified portion of the screen into a Windows bitmap file.

Prototype

INTERFACE
    INTEGER(4) FUNCTION SAVEIMAGE(FNAME, X1, Y1, X2, Y2)
    CHARACTER(LEN=*) FNAME
    INTEGER(4) X1, Y1, X2, Y2
END FUNCTION
END INTERFACE

FNAME Input. CHARACTER(LEN=*) Path of the bitmap file.
X1, Y1 Input. INTEGER(4) Viewport coordinates for upper-left corner of the screen image to be captured.
X2, Y2 Input. INTEGER(4) Viewport coordinates for lower-right corner of the screen image to be captured.

Description

The SAVEIMAGE function captures the screen image within a rectangle defined by the upper-left and lower-right screen coordinates and stores the image as a Windows bitmap file specified by FNAME. The image is stored with a palette containing the colors displayed on the screen. SAVEIMAGE defines the bounding rectangle in viewport coordinates.

Output

The result type is INTEGER(4). The result is zero if successful; otherwise, a negative value.
SAVEIMAGE_W

Saves an image from a specified portion of the screen into a Windows bitmap file.

Prototype

INTERFACE
  INTEGER(4) FUNCTION SAVEIMAGE_W(FNAME,WX1, WY1, WX2, WY2)
    CHARACTER(LEN=*) FNAME
    DOUBLE PRECISION WX1, WY1, WX2, WY2
  END FUNCTION
END INTERFACE

FNAME  Input. CHARACTER(LEN=*). Path of the bitmap file.
WX1, WY1  Input. REAL(8). Window coordinates for upper-left corner of the screen image to be captured.
WX2, WY2  Input. REAL(8). Window coordinates for lower-right corner of the screen image to be captured.

Description

The SAVEIMAGE_W function captures the screen image within a rectangle defined by the upper-left and lower-right screen coordinates and stores the image as a Windows bitmap file specified by FNAME. The image is stored with a palette containing the colors displayed on the screen. SAVEIMAGE_W defines the bounding rectangle in window coordinates.

Output

The result type is INTEGER(4). The result is zero if successful; otherwise, a negative value.
SAVEJPEG

Saves an image from a specified portion of the screen into a JPEG graphic file.

Prototype

INTERFACE
    INTEGER(4) FUNCTION SAVEJPEG(FNAME, X1, Y1, X2, Y2, JPGQUALITY)
    CHARACTER(LEN=*) FNAME
    INTEGER(4) X1, Y1, X2, Y2, JPGQUALITY
END FUNCTION

END INTERFACE

FNAME  Input. CHARACTER (LEN=*)  . Path of the bitmap file.
X1, Y1  Input. INTEGER (4)  . Viewport coordinates for upper-left corner of the screen image to be captured.
X2, Y2  Input. INTEGER (4)  . Viewport coordinates for lower-right corner of the screen image to be captured.
JPGQUALITY  Input. INTEGER (4)  . Quality of stored JPEG image. [0-100] where highest quality is 100.

Description

The SAVEJPEG function captures the screen image within a rectangle defined by the upper-left and lower-right screen coordinates and stores the image as a JPEG graphic file specified by FNAME.

SAVEJPEG defines the bounding rectangle in viewport coordinates.

Output

The result type is INTEGER (4). The result is zero if successful; otherwise, a negative value.
SAVEJPEG_W

Saves an image from a specified portion of the screen into a JPEG graphic file.

Prototype

INTERFACE

   INTEGER(4) FUNCTION SAVEJPEG_W(FNAME, WX1, WY1, WX2, WY2, JPGWQUALITY)
   CHARACTER(LEN=*) FNAME
   DOUBLE PRECISION WX1, WY1, WX2, WY2, WJPGQUALITY
END FUNCTION
END INTERFACE

FNAME     Input. CHARACTER (LEN=*) . Path of the bitmap file.
WX1, WY1  Input. REAL(8) . Viewport coordinates for upper-left corner of the screen image to be captured.
WX2, WY2  Input. REAL(8) . Viewport coordinates for lower-right corner of the screen image to be captured.
JPGWQUALITY Input. INTEGER(4) . Quality of stored JPEG image. [0-100] where highest quality is 100.

Description

The SAVEJPEG_W function captures the screen image within a rectangle defined by the upper-left and lower-right screen coordinates and stores the image as a JPEG graphic file specified by FNAME.

SAVEJPEG_W defines the bounding rectangle in window coordinates

Output

The result type is INTEGER(4) . The result is zero if successful; otherwise, a negative value.
SCROLLTEXTWINDOW

Scrolls the contents of a text window.

Prototype

INTERFACE
  SUBROUTINE SCROLLTEXTWINDOW(ROWS)
    INTEGER(2) ROWS
  END SUBROUTINE
END INTERFACE

ROWS       Input. INTEGER(2). Number of rows to scroll.

Description

The SCROLLTEXTWINDOW subroutine scrolls the text in a text window (previously defined by SETTEXTWINDOW). The default text window is the entire window.

The rows argument specifies the number of lines to scroll. A positive value for rows scrolls the window up (the usual direction); a negative value scrolls the window down. Specifying a number larger than the height of the current text window is equivalent to calling CLEARSCREEN ($GWINDOW). A value of 0 for ROWS has no effect.
SETBKCOLOR

Sets the current background color index for both text and graphics.

Prototype

```plaintext
INTERFACE
  FUNCTION SETBKCOLOR(COLOR)
    INTEGER(4) SETBKCOLOR, COLOR
  END FUNCTION
END INTERFACE
```

COLOR  
Input. INTEGER(4). Color index to set the background color to.

Description

SETBKCOLOR changes the background color index for both text and graphics. The color index of text over the background color is set with SETTEXTCOLOR. The color index of graphics over the background color (used by drawing functions such as FLOODFILL and ELLIPSE) is set with SETCOLOR. These non-RGB color functions use color indexes, not true color values, and limit the user to colors in the palette, at most 256. For access to all system colors, use SETBKCOLORRGB, SETCOLORRGB, and SETTEXTCOLORRGB.

Changing the background color index does not change the screen immediately. The change becomes effective when CLEARSCREEN is executed or when doing text input or output, such as with READ, WRITE, or OUTTEXT. The graphics output function OUTGTEXT does not affect the color of the background.

Generally, INTEGER(4) color arguments refer to color values and INTEGER(2) color arguments refer to color indexes. The two exceptions are GETBKCOLOR and SETBKCOLOR. The default background color index is 0, which is associated with black unless the user remaps the palette with REMAPPALETTERGB.
Output

The result type is INTEGER(4). The result is the previous background color index.

SETCLIPRGN

Limits graphics output to part of the screen.

Prototype

INTERFACE
    SUBROUTINE SETCLIPRGN(X1,Y1,X2,Y2)
        INTEGER(2) X1,Y1,X2,Y2
    END SUBROUTINE
END INTERFACE

X1, Y1  Input. INTEGER(2). Physical coordinates for upper-left corner of clipping region.
X2, Y2  Input. INTEGER(2). Physical coordinates for lower-right corner of clipping region.

Description

The SETCLIPRGN function limits the display of subsequent graphics output and font text output to that which fits within a designated area of the screen (the "clipping region"). The physical coordinates (X1, Y1) and (X2, Y2) are the upper-left and lower-right corners of the rectangle that defines the clipping region. The SETCLIPRGN function does not change the viewport-coordinate system; it merely masks graphics output to the screen.

SETCLIPRGN affects graphics and font text output only, such as OUTGTEXT. To mask the screen for text output using OUTTEXT, use SETTEXTWINDOW.
SETCOLOR

Sets the current graphics color index.

Prototype

INTERFACE
FUNCTION SETCOLOR(COLOR)
INTEGER(2) SETCOLOR
INTEGER(2) COLOR
END FUNCTION
END INTERFACE

COLOR Input. INTEGER(2). Color index to set the current graphics color to.

Description

The SETCOLOR function sets the current graphics color index, which is used by graphics functions such as ELLIPSE. The background color index is set with SETBKCOLOR. The color index of text over the background color is set with SETTEXTCOLOR. These non-RGB color functions use color indexes, not true color values, and limit the user to colors in the palette, at most 256. For access to all system colors, use SETCOLORRGB, SETBKCOLORRGB, and SETTEXTCOLORRGB.

Output

The result type is INTEGER(2). The result is the previous color index if successful; otherwise, -1.
SETFILLMASK

Sets the current fill mask to a new pattern.

Prototype

```
INTERFACE
  SUBROUTINE SETFILLMASK(MASK)
    INTEGER(1) MASK(8)
  END SUBROUTINE
END INTERFACE
```

MASK
Input. INTEGER(1). One-dimensional array of length 8.

Description

There are 8 bytes in MASK, and each of the 8 bits in each byte represents a pixel, creating an 8x8 pattern. The first element (byte) of MASK becomes the top 8 bits of the pattern, and the eighth element (byte) of MASK becomes the bottom 8 bits.

During a fill operation, pixels with a bit value of 1 are set to the current graphics color, while pixels with a bit value of zero are set to the current background color. The current graphics color is set with SETCOLORRGB or SETCOLOR. The 8-byte mask is replicated over the entire fill area. If no fill mask is set (with SETFILLMASK), or if the mask is all ones, solid current color is used in fill operations.

The fill mask controls the fill pattern for graphics routines (FLOODFILLRGB, PIE, ELLIPSE, POLYGON, and RECTANGLE).

To change the current fill mask, determine the array of bytes that corresponds to the desired bit pattern and set the pattern with SETFILLMASK, as in the following example.
SETLINESTYLE

Sets the current line style to a new line style.

Prototype

INTERFACE
SUBROUTINE SETLINESTYLE(MASK)
INTEGER(2) MASK
END SUBROUTINE
END INTERFACE

MASK Input. INTEGER(2). Desired Quickwin line-style mask. (See the table below.)

Description

The mask is mapped to the style that most closely equates the percentage of the bits in the mask that are set. The style produces lines that cover a certain percentage of the pixels in that line.

SETLINESTYLE sets the style used in drawing a line. You can choose from the following styles:

<table>
<thead>
<tr>
<th>QuickWin Mask</th>
<th>Internal Windows Style</th>
<th>Selection Criteria</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>PS_SOLID</td>
<td>16 bits on</td>
<td>__________</td>
</tr>
<tr>
<td>0xEEEE</td>
<td>PS_DASH</td>
<td>11 to 15 bits on</td>
<td>-----------</td>
</tr>
<tr>
<td>0xECEC</td>
<td>PS_DASHDOT</td>
<td>10 bits on</td>
<td>~~ ~ ~ ~ ~ ~</td>
</tr>
<tr>
<td>0xECCE</td>
<td>PS_DASHDOTDOT</td>
<td>9 bits on</td>
<td>~ ~ ~ ~ ~ ~</td>
</tr>
<tr>
<td>0xAAAA</td>
<td>PS_DOT</td>
<td>1 to 8 bits on</td>
<td>................</td>
</tr>
<tr>
<td>0x0000</td>
<td>PS_NULL</td>
<td>0 bits on</td>
<td>...............</td>
</tr>
</tbody>
</table>
SETLINESTYLE affects the drawing of straight lines as in LINETO, POLYGON, and RECTANGLE, but not the drawing of curved lines as in ARC, ELLIPSE, or PIE.

The current graphics color is set with SETCOLORGB or SETCOLOR. SETWRITEMODE affects how the line is displayed.

SETPIXEL

Sets a pixel at a specified location to the current graphics color index.

Prototype

```fortran
INTERFACE
  FUNCTION SETPIXEL(X, Y)
    INTEGER(2) SETPIXEL, X, Y
  END FUNCTION
END INTERFACE
```

X, Y  Input. INTEGER(2). Viewport coordinates for target pixel.

Description

SETPIXEL sets the specified pixel to the current graphics color index. The current graphics color index is set with SETCOLOR and retrieved with GETCOLOR. The non-RGB color functions (such as SETCOLOR and SETPIXELS) use color indexes rather than true color values.

Output

The result type is INTEGER(2). The result is the previous color index of the target pixel if successful; otherwise, -1 (for example, if the pixel lies outside the clipping region).
**SETPIXEL_W**

*Sets a pixel at a specified location to the current graphics color index.*

**Prototype**

```fortran
INTERFACE
    FUNCTION SETPIXEL_W(WX, WY)
        INTEGER(2) SETPIXEL_W
        DOUBLE PRECISION WX, WY
    END FUNCTION
END INTERFACE
```

**Input.** REAL(8). Window coordinates for target pixel.

**Description**

SETPIXEL_W sets the specified pixel to the current graphics color index. The current graphics color index is set with SETCOLOR and retrieved with GETCOLOR. The non-RGB color functions (such as SETCOLOR and SETPIXELS) use color indexes rather than true color values.

**Output**

The result type is INTEGER(2). The result is the previous color index of the target pixel if successful; otherwise, -1 (for example, if the pixel lies outside the clipping region).
SETPIXELS

Sets the color indexes of multiple pixels.

Prototype

INTERFACE
SUBROUTINE SETPIXELS (N, X, Y, COLOR)
  INTEGER (4) N ! size of arrays
  INTEGER (2) X(*), Y(*) ! x, y coordinates
  INTEGER (2) COLOR(*) ! palette indices
END SUBROUTINE
END INTERFACE

N
Input. INTEGER (4). Number of pixels to set. Sets the number of elements in the other arguments.

X, Y
Input. INTEGER (2). Parallel arrays containing viewport coordinates of pixels to set.

COLOR
Input. INTEGER (2). Array containing color indexes to set the pixels to.

Description

SETPIXELS sets the pixels specified in the arrays \( x \) and \( y \) to the color indexes in \( \text{COLOR} \). These arrays are parallel: the first element in each of the three arrays refers to a single pixel, the second element refers to the next pixel, and so on.

If any of the pixels are outside the clipping region, those pixels are ignored. Calls to SETPIXELS with \( N \) less than 1 are also ignored. SETPIXELS is a much faster way to set multiple pixel color indexes than individual calls to SETPIXEL.

Unlike SETPIXELS, SETPIXELSRGB gives access to the full color capacity of the system by using direct color values rather than indexes to a palette. The non-RGB color functions (such as SETPIXELS and SETCOLOR) use color indexes rather than true color values.
If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

### SETTEXTCOLOR

*Sets the current text color index.*

**Prototype**

```plaintext
INTERFACE
FUNCTION SETTEXTCOLOR (INDEX)
    INTEGER (2) SETTEXTCOLOR, INDEX
END FUNCTION
END INTERFACE
```

**INDEX**

Input. INTEGER (2). Color index to set the text color to.

**Description**

`SETTEXTCOLOR` sets the current text color index. The default value is 15, which is associated with white unless the user remaps the palette.

`GETTEXTCOLOR` returns the text color index set by `SETTEXTCOLOR`.

`SETTEXTCOLOR` affects text output with `OUTTEXT`, `WRITE`, and `PRINT`.

**Output**

The result type is INTEGER (2). The result is the previous text color index.
**SETTEXTPOSITION**

Sets the current text position to a specified position relative to the current text window.

### Prototype

```fortran
INTERFACE
SUBROUTINE SETTEXTPOSITION(ROW, COL, S)
   INTEGER(2) ROW, COL
   integer*2 col
   STRUCTURE /RCCOORD/
      INTEGER(2) ROW, COL
      integer*2 col
   END STRUCTURE
   RECORD /RCCOORD/S
END SUBROUTINE
END INTERFACE
```

**ROW**
Input. INTEGER(2). New text row position.

**COL**
Input. INTEGER(2). New text column position.

**S**
Output. RECORD /XYCOORD/. Previous text position.

### Description

Subsequent text output with the OUTTEXT function (as well as standard console I/O statements, such as PRINT and WRITE) begins at the point (ROW, COL).
**SETEXTWINDOW**

Sets the current text window.

**Prototype**

```plaintext
INTERFACE
  SUBROUTINE SETEXTWINDOW(R1,C1,R2,C2)
    INTEGER(2) R1,C1,R2,C2
  END SUBROUTINE
END INTERFACE
```

- **R1,C1** Input. INTEGER(2). Row and column coordinates for upper-left corner of the text window.
- **R2,C2** Input. INTEGER(2). Row and column coordinates for lower-right corner of the text window.

**Description**

SETEXTWINDOW specifies a window in row and column coordinates where text output to the screen using OUTTEXT, WRITE, or PRINT will be displayed. You set the text location within this window with SETTEXTPOSITION.

Text is output from the top of the window down. When the window is full, successive lines overwrite the last line.

SETEXTWINDOW does not affect the output of the graphics text routine OUTGTEXT. Use the SETVIEWPORT function to control the display area for graphics output.
SETVIEWORG

Moves the viewport-coordinate origin (0, 0) to the specified physical point.

Prototype

INTERFACE
SUBROUTINE SETVIEWORG(X,Y,S)
INTEGER(2) X,Y
integer*2 y
STRUCTURE /XYCOORD/
    INTEGER(2) XCOORD,YCOORD
END STRUCTURE
RECORD /XYCOORD/s
END SUBROUTINE
END INTERFACE

\(X, Y\)  Input. \text{INTEGER(2)}. Physical coordinates of new viewport origin.

\(S\)  Output. \text{RECORD /XYCOORD/}. Physical coordinates of the previous viewport origin.

Description

The \text{XYCOORD} type variable \(S\), returns the physical coordinates of the previous viewport origin.
SETVIEWPORT

Redefines the graphics viewport

Prototype

```fortran
INTERFACE
  SUBROUTINE SETVIEWPORT(X1,Y1,X2,Y2)
    INTEGER(2) X1,Y1,X2,Y2
  END SUBROUTINE
END INTERFACE
```

`X1,Y1` Input. INTEGER(2). Physical coordinates for upper-left corner of viewport.

`X2,Y2` Input. INTEGER(2). Physical coordinates for lower-right corner of viewport.

Description

Redefines the graphics viewport by defining a clipping region in the same manner as SETCLIPRGN and then setting the viewport-coordinate origin to the upper-left corner of the region. The physical coordinates `(X1, Y1)` and `(X2, Y2)` are the upper-left and lower-right corners of the rectangular clipping region. Any window transformation done with the SETWINDOW function is relative to the viewport, not the entire screen.
SETWINDOW

Defines a window bound by the specified coordinates.

Prototype

INTERFACE
    FUNCTION SETWINDOW(FINVERT, WX1, WY1, WX2, WY2)
        INTEGER(2) SETWINDOW
        LOGICAL(2) FINVERT
        DOUBLE PRECISION WX1, WY1, WX2, WY
    END FUNCTION
END INTERFACE

FINVERT Input. LOGICAL(2). Direction of increase of the y-axis. If FINVERT is .TRUE., the y-axis increases from the window bottom to the window top (as Cartesian coordinates). If FINVERT is .FALSE., the y-axis increases from the window top to the window bottom (as pixel coordinates).

WX1, WY1 Input. REAL(8). Window coordinates for upper-left corner of window.

WX2, WY2 Input. REAL(8). Window coordinates for lower-right corner of window.

Description

The SETWINDOW function determines the coordinate system used by all window-relative graphics routines. Any graphics routines that end in _W (such as ARC_W, RECTANGLE_W, and LINETO_W) use the coordinate system set by SETWINDOW.

Any window transformation done with the SETWINDOW function is relative to the viewport, not the entire screen.
An arc drawn using inverted window coordinates is not an upside-down version of an arc drawn with the same parameters in a noninverted window. The arc is still drawn counterclockwise, but the points that define where the arc begins and ends are inverted.

If \( WX_1 \) equals \( WX_2 \) or \( WY_1 \) equals \( WY_2 \), \texttt{SETWINDOW} fails.

**Output**

The result type is \texttt{INTEGER(2)}. The result is The result is nonzero if successful; otherwise, 0 (for example, if the program that calls \texttt{SETWINDOW} is not in a graphics mode).

---

**SETWRITEMODE**

*Sets the current logical write mode.*

Prototype

```plaintext
INTERFACE
    FUNCTION SETWRITEMODE(WMODE)
        INTEGER(2) SETWRITEMODE, WMODE
    END FUNCTION
END INTERFACE
```

\( WMODE \) Input. INTEGER(2). Write mode to be set. One of the following symbolic constants:

- \$GPSET Causes lines to be drawn in the current graphics color. (Default)
- \$GAND Causes lines to be drawn in the color that is the logical AND of the current graphics color and the current background color.
- \$GOR Causes lines to be drawn in the color that is the logical OR of the current graphics color and the current background color.
$GPRESET  Causes lines to be drawn in the color that is the logical NOT of the current graphics color.

$GXOR  Causes lines to be drawn in the color that is the logical exclusive OR (XOR) of the current graphics color and the current background color.

In addition, one of the following binary raster operation constants can be used (described in the online documentation for the WIN32 API SetROP2):

$GR2_BLACK
$GR2_NOTMERGEPEN
$GR2_MASKNOTPEN
$GR2_NOTCOPYPEN  (same as $GPRESET)
$GR2_MASKPENNOT
$GR2_NOT
$GR2_XORPEN  (same as $GXOR)
$GR2_NOTMASKPEN
$GR2_MASKPEN  (same as $GAND)
$GR2_NOTXORPEN
$GR2_NOP
$GR2_MERGENOTPEN
$GR2_COPYPEN  (same as $GPSET)
$GR2_MERGEPENNOT
$GR2_MERGEPEN  (same as $GOR)
$GR2_WHITE

**Description**

Sets the current logical write mode, which is used when drawing lines with the LINE TO, POLYGON, and RECTANGLE functions. The current graphics color is set with SETCOLORRGB (or SETCOLOR) and the current
background color is set with \texttt{SETBKCOLORRGB} (or \texttt{SETBKCOLOR}). As an example, suppose you set the background color to yellow (\#00FFFF) and the graphics color to purple (\#FF00FF) with the following commands:

\begin{verbatim}
OLDCOLOR = SETBKCOLORRGB(#00FFFF)
CALL CLEARSCREEN($GCLEARSCREEN)
OLDCOLOR = SETCOLORRGB(#FF00FF)
\end{verbatim}

If you then set the write mode with the \texttt{$GAND} option, lines are drawn in red (\#0000FF); with the \texttt{$GOR} option, lines are drawn in white (\#FFFFFF); with the \texttt{$GXOR} option, lines are drawn in turquoise (\#FFFF00); and with the \texttt{$GRESET} option, lines are drawn in green (\#00FF00). Setting the write mode to \texttt{$GPSET} causes lines to be drawn in the graphics color.

\textbf{Output}

The result type is \texttt{INTEGER(2)}. The result is the previous write mode if successful; otherwise, -1.

\section*{WRAPON}

\textit{Controls the text output wrap.}

\section*{Prototype}

\begin{verbatim}
INTERFACE
  FUNCTION WRAPON(OPTION)
    INTEGER(2) WRAPON,OPTION
  END FUNCTION
END INTERFACE
\end{verbatim}

\begin{itemize}
  \item \textbf{OPTION} \hspace{0.5cm} Input. \texttt{INTEGER(2)}. Wrap mode. One of the following symbolic constants:
    \begin{itemize}
      \item \texttt{$GWRAPOFF$} \hspace{0.5cm} Truncates lines at right edge of window border.
      \item \texttt{$GWRAPON$} \hspace{0.5cm} Wraps lines at window border, scrolling if necessary.
    \end{itemize}
\end{itemize}
Description
Controls whether text output with the OUTTEXT function wraps to a new line or is truncated when the text output reaches the edge of the defined text window. WRAPON does not affect font routines such as OUTGTEXT.

Output
The result type is INTEGER(2). The result is the previous value of OPTION.

Per Pixel and Color Functions

GETCOLORRGB
Gets the current graphics color
Red-Green-Blue (RGB) value.

Prototype
INTERFACE
  FUNCTION GETCOLORRGB()
    integer*4 GETCOLORRGB
  END FUNCTION
END INTERFACE

Description
Gets the current graphics color Red-Green-Blue (RGB) value (used by graphics functions such as ARC, ELLIPSE, and FLOODFILLRGB). In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you retrieve with GETCOLORRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:
Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

GETCOLORRGB returns the RGB color value of graphics over the background color (used by graphics functions such as ARC, ELLIPSE, and FLOODFILLRGB), set with SETCOLORRGB. GETBCKCOLORRGB returns the RGB color value of the current background for both text and graphics, set with SETBCKCOLORRGB. GETTEXTCOLORRGB returns the RGB color value of text over the background color (used by text functions such as OUTTEXT, WRITE, and PRINT), set with SETTEXTCOLORRGB.

SETCOLORRGB (and the other RGB color selection functions SETBCKCOLORRGB and SETTEXTCOLORRGB) sets the color to a value chosen from the entire available range. The non-RGB color functions (SETCOLOR, SETBCKCOLOR, and SETTEXTCOLOR) use color indexes rather than true color values. If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

**Output**

The result type is INTEGER(4). The result is the RGB value of the current graphics color.
GETBKCOLORRGB

Gets the current background Red-Green-Blue (RGB) color value for both text and graphics.

Prototype

INTERFACE
    FUNCTION GETBKCOLORRGB ()
        INTEGER(4) GETBKCOLORRGB
    END FUNCTION
END INTERFACE

Description

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you retrieve with GETBKCOLORRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

GETBKCOLORRGB returns the RGB color value of the current background for both text and graphics, set with SETBKCOLORRGB. The RGB color value of text over the background color (used by text functions such as OUTTEXT, WRITE, and PRINT) is set with SETTEXTCOLORRGB and returned with GETTEXTCOLORRGB. The RGB color value of graphics over the background color (used by graphics functions such as ARC, OUTGTEXT, and FLOODFILLRGB) is set with SETCOLORRGB and returned with GETCOLORRGB.
SETBKCOLORRGB (and the other RGB color selection functions
SETCOLORRGB and SETTEXTCOLORRGB) sets the color to a value chosen from the entire available range. The non-RGB color functions
(SETBKCOLOR, SETCOLOR, and SETTEXTCOLOR) use color indexes rather than true color values. If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

Output
The result type is INTEGER (4). The result is the RGB value of the current background color for both text and graphics.

GETPIXELRGB

Returns the Red-Green-Blue (RGB) color value of the pixel at a specified location.

Prototype

```
INTERFACE
   FUNCTION GETPIXELRGB(X,Y)
       INTEGER(4) GETPIXELRGB
       INTEGER(2) X,Y
   END FUNCTION
END INTERFACE
```

X, Y       Input. INTEGER (2). Viewport coordinates for pixel position.
Description

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you retrieve with GETPIXELRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

GETPIXELRGB returns the true color value of the pixel, set with SETPIXELRGB, SETCOLORRGB, SETBKCOLORRGB, or SETTEXTCOLORRGB, depending on the pixel's position and the current configuration of the screen.

SETPIXELRGB (and the other RGB color selection functions SETCOLORRGB, SETBKCOLORRGB, and SETTEXTCOLORRGB) sets colors to a color value chosen from the entire available range. The non-RGB color functions (SETPIXELS, SETCOLOR, SETBKCOLOR, and SETTEXTCOLOR) use color indexes rather than true color values. If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit Red-Green-Blue (RGB) value with an RGB color function, rather than a palette index with a non-RGB color function.

Output

The result type is INTEGER (4). The result is the pixel's current RGB color value.
**GETPIXELRGB_W**

*Returns the Red-Green-Blue (RGB) color value of the pixel at a specified location.*

**Prototype**

```fortran
INTERFACE
  FUNCTION GETPIXELRGB_W(WX, WY)
    INTEGER(4) GETPIXELRGB_W
    REAL*8 WX, WY
    REAL*8 wy
  END FUNCTION
END INTERFACE
```

**Description**

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you retrieve with **GETPIXELRGB_W**, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 11111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

**GETPIXELRGB_W** returns the true color value of the pixel, set with **SETPIXELRGB_W**, **SETCOLORRGB**, **SETBKCOLORRGB**, or **SETTEXTCOLORRGB**, depending on the pixel's position and the current configuration of the screen.
SETPIXELRGB (and the other RGB color selection functions
SETCOLORRGB, SETBKCOLORRGB, and SETTEXTCOLORRGB) sets colors
to a color value chosen from the entire available range. The non-RGB color
functions (SETPIXELS, SETCOLOR, SETBKCOLOR, and SETTEXTCOLOR)
use color indexes rather than true color values. If you use color indexes, you
are limited to the colors available in the palette, at most 256. Some display
adapters (SVGA and true color) are capable of creating 262,144 (256K)
colors or more. To access any available color, you need to specify an
explicit Red-Green-Blue (RGB) value with an RGB color function, rather
than a palette index with a non-RGB color function.

Output
The result type is INTEGER(4). The result is the pixel's current RGB color
value.

SETPIXELSRGB
Sets multiple pixels to the given
Red-Green-Blue (RGB) color.

Prototype
INTERFACE
  SUBROUTINE SETPIXELSRGB (N, X, Y, COLOR)
    INTEGER(4) N
    INTEGER(2) X(*), Y(*)
    INTEGER(4) COLOR(*)
  END SUBROUTINE
END INTERFACE

N      Input. INTEGER(4). Number of pixels to be changed.
      Determines the number of elements in arrays x and y.
X, Y   Input. INTEGER(2). Parallel arrays containing
      viewport coordinates of the pixels to set.
**COLOR**

Input. **INTEGER (4)**. Array containing the RGB color values to set the pixels to. Range and result depend on the system's display adapter.

**Description**

**SETPIXELSRGB** sets the pixels specified in the arrays x and y to the RGB color values in **COLOR**. These arrays are parallel: the first element in each of the three arrays refers to a single pixel, the second element refers to the next pixel, and so on.

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you set with **SETPIXELSRGB**, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 11111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

A good use for **SETPIXELSRGB** is as a buffering form of **SETPIXELRGB**, which can improve performance substantially. The example code shows how to do this.

If any of the pixels are outside the clipping region, those pixels are ignored. Calls to **SETPIXELSRGB** with n less than 1 are also ignored.

**SETPIXELSRGB** (and the other RGB color selection functions such as **SETPIXELRGB** and **SETCOLORRGB**) sets colors to values chosen from the entire available range. The non-RGB color functions (such as **SETPIXELS** and **SETCOLOR**) use color indexes rather than true color values.

If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.
GETPIXELSRGB

Returns the Red-Green-Blue (RGB) color values of multiple pixels.

Prototype

```fortran
INTERFACE
SUBROUTINE GETPIXELSRGB(N, X, Y, COLOR)
    INTEGER(4) N
    INTEGER(2) X(*), Y(*)
    INTEGER(4) COLOR (*)
END SUBROUTINE
END INTERFACE
```

N
Input. INTEGER (4). Number of pixels to get. Sets the number of elements in the other argument arrays.

X, Y
Input. INTEGER (2). Parallel arrays containing viewport coordinates of pixels.

COLOR
Output. INTEGER (4). Array to be filled with RGB color values of the pixels at x and y.

Description

GETPIXELS fills in the array COLOR with the RGB color values of the pixels specified by the two input arrays x and y. These arrays are parallel: the first element in each of the three arrays refers to a single pixel, the second element refers to the next pixel, and so on.

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you retrieve with GETPIXELSRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:
Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

GETPIXELSRGB is a much faster way to acquire multiple pixel RGB colors than individual calls to GETPIXELRGB. GETPIXELSRGB returns an array of true color values of multiple pixels, set with SETPIXELSRGB, SETCOLORRGB, SETBKCOLORRGB, or SETTEXTCOLORRGB, depending on the pixels' positions and the current configuration of the screen.

SETPIXELSRGB (and the other RGB color selection functions SETCOLORRGB, SETBKCOLORRGB, and SETTEXTCOLORRGB) sets colors to a color value chosen from the entire available range. The non-RGB color functions (SETPIXELS, SETCOLOR, SETBKCOLOR, and SETTEXTCOLOR) use color indexes rather than true color values. If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

**SETCOLORRGB**

*Sets the current graphics color to the specified Red-Green-Blue (RGB) value.*

**Prototype**

```plaintext
INTERFACE
   FUNCTION SETCOLORRGB(COLOR)
       INTEGER(4) SETCOLORRGB,COLOR
   END FUNCTION
END INTERFACE
```
COLOR Input. INTEGER (4). RGB color value to set the current graphics color to. Range and result depend on the system's display adapter.

Description
In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you specify with SETCOLORRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

SETCOLORRGB sets the RGB color value of graphics over the background color, used by the following graphics functions: ARC, ELLIPSE, FLOODFILL, LINETO, OUTGTEXT, PIE, POLYGON, RECTANGLE, and SETPIXEL. SETBKCOLORRGB sets the RGB color value of the current background for both text and graphics. SETTEXTCOLORRGB sets the RGB color value of text over the background color (used by text functions such as OUTTEXT, WRITE, and PRINT).

SETCOLORRGB (and the other RGB color selection functions SETBKCOLORRGB, and SETTEXTCOLORRGB) sets the color to a value chosen from the entire available range. The non-RGB color functions (SETCOLOR, SETBKCOLOR, and SETTEXTCOLOR) use color indexes rather than true color values. If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

Output
The result type is INTEGER (4). The result is the previous RGB color value.
SETBKCOLORRGB

Sets the current background color to the given Red-Green-Blue (RGB) value.

Prototype

INTERFACE
  FUNCTION SETBKCOLORRGB(color)
    INTEGER(4) SETBKCOLORRGB, COLOR
  END FUNCTION
END INTERFACE

COLOR Input. INTEGER(4). RGB color value to set the background color to. Range and result depend on the system's display adapter.

Description

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you specify with SETBKCOLORRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

SETBKCOLORRGB sets the RGB color value of the current background for both text and graphics. The RGB color value of text over the background color (used by text functions such as OUTTEXT, WRITE, and PRINT) is set with SETTEXTCOLORRGB. The RGB color value of graphics over the background color (used by graphics functions such as ARC, OUTGTEXT, and FLOODFILLRGB) is set with SETCOLORRGB.
SETBKCOLORRGB (and the other RGB color selection functions SETCOLORRGB, and SETTEXTCOLORRGB) sets the color to a value chosen from the entire available range. The non-RGB color functions (SETCOLOR, SETBKCOLOR, and SETTEXTCOLOR) use color indexes rather than true color values. If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color.

**Output**

The result type is INTEGER(4). The result is the previous background RGB color value.

---

**SETPIXELRGB**

*Sets a pixel at a specified location to the specified Red-Green-Blue (RGB) color value.*

**Prototype**

```fortran
INTERFACE
  FUNCTION SETPIXELRGB(X,Y,COLOR)
    INTEGER(4) SETPIXELRGB, COLOR
    INTEGER(2) X, Y
  END FUNCTION
END INTERFACE
```

**X, Y**

Input. INTEGER(2). Viewport coordinates for target pixel.

**COLOR**

Input. INTEGER(4). RGB color value to set the pixel to. Range and result depend on the system’s display adapter.
Description
In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you specify with SETPIXELRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:
Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

SETPIXELRGB (and the other RGB color selection functions such as SETPIXELSRGB, SETCOLORRGB) sets the color to a value chosen from the entire available range. The non-RGB color functions (such as SETPIXELS and SETCOLOR) use color indexes rather than true color values.

If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

Output
The result type is INTEGER (4). The result is the previous RGB color value of the pixel.
**SETPIXELRGB_W**

Sets a pixel at a specified location to the specified Red-Green-Blue (RGB) color value.

### Prototype

```fortran
INTERFACE
FUNCTION SETPIXELRGB_W(X, Y, COLOR)
  INTEGER(4)  SETPIXELRGB_W, COLOR
  REAL*8     WX, WY
END FUNCTION
END INTERFACE
```

- **WX, WY**
  - Input. REAL(8). Window coordinates for target pixel.
- **COLOR**
  - Input. INTEGER(4). RGB color value to set the pixel to. Range and result depend on the system's display adapter.

### Description

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you specify with `SETPIXELRGB_W`, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

`SETPIXELRGB_W` (and the other RGB color selection functions such as `SETPIXELSRGB`, `SETCOLORRGB`) sets the color to a value chosen from the entire available range. The non-RGB color functions (such as `SETPIXELS` and `SETCOLOR`) use color indexes rather than true color values.
If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

**Output**
The result type is `INTEGER (4)`. The result is the previous RGB color value of the pixel.

---

**RGBTOINTEGER**

Converts three integers specifying red, green, and blue color intensities into a four-byte RGB integer.

**Prototype**

```fortran
INTERFACE
  FUNCTION RGBTOINTEGER(RED, GREEN, BLUE)
    INTEGER(4) RGBTOINTEGER, RED, GREEN, BLUE
  END FUNCTION
END INTERFACE
```

- **RED**
  Input. `INTEGER (4)`. Intensity of the red component of the RGB color value. Only the lower 8 bits of `RED` are used.

- **GREEN**
  Input. `INTEGER (4)`. Intensity of the green component of the RGB color value. Only the lower 8 bits of `GREEN` are used.

- **BLUE**
  Input. `INTEGER (4)`. Intensity of the blue component of the RGB color value. Only the lower 8 bits of `BLUE` are used.
Description
Converts three integers specifying red, green, and blue color intensities into a four-byte RGB integer for use with RGB functions and subroutines. In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value returned withRGBTOINTEGER, red is the rightmost byte, followed by green and blue. The RGB value’s internal structure is as follows:
Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

Output
The result type is INTEGER(4). The result is the combined RGB color value.

INTERGERTORGB
Converts an RGB color value into its red, green, and blue components.

Prototype

INTERFACE
  SUBROUTINE INTEGER(4) (RGB, RED, GREEN, BLUE)
    INTEGER(4) RGB, RED, GREEN, BLUE
  END SUBROUTINE
END INTERFACE

RGB Input. INTEGER(4). RGB color value whose red, green, and blue components are to be returned.

RED Output. INTEGER(4). Intensity of the red component of the RGB color value.
GREEN Output. INTEGER(4). Intensity of the green component of the RGB color value.

BLUE Output. INTEGER(4). Intensity of the blue component of the RGB color value.

**Description**

`INTEGRERTORGB` separates the four-byte RGB color value into the three components as follows:

**Font Manipulation Functions**

**GETFONTINFO**

*Gets the current font characteristics.*

**Prototype**

```fortran
INTERFACE
FUNCTION GETFONTINFO(FI)
INTEGER(2) GETFONTINFO
STRUCTURE /FONTINFO/
  INTEGER(4) TYPE
  INTEGER(4) ASCENT
  INTEGER(4) PIXWIDTH
  INTEGER(4) PIXHEIGHT
  INTEGER(4) AVGWIDTH
  CHARACTER(LEN=81) FILENAME
  CHARACTER(LEN=32) FACENAME
  LOGICAL(1) ITALIC
  LOGICAL(1) EMPHASIZED
  LOGICAL(1) UNDELINE
END STRUCTURE
```
RECORD /FONTINFO/ FI
END FUNCTION
END INTERFACE

Output. RECORD /FONTINFO/. Set of characteristics of the current font.

STRUCTURE /FONTINFO/
  INTEGER(4) TYPE ! 1 = truetype, 0 = bit map
  INTEGER(4) ASCENT ! Pixel distance from top to baseline
  INTEGER(4) PIXWIDTH ! Character width in pixels, 0 = proportional
  INTEGER(4) PIXHEIGHT ! Character height in pixels
  INTEGER(4) AVGWIDTH ! Average character width in pixels
  CHARACTER (81) ! File name
  CHARACTER(32) FACENAME ! Font name
  LOGICAL(1) ITALIC ! .TRUE. if current font formatted italic
  LOGICAL(1) EMPHASIZED ! .TRUE. if current font formatted bold
  LOGICAL(1) UNDERLINE ! .TRUE. if current font formatted underlined
END STRUCTURE

Description
You must initialize fonts with INITIALIZEFONTS before calling any font-related function, including GETFONTINFO.

Output
The result type is INTEGER (2). The result is zero if successful; otherwise, -1.
GETGTEXTEXTENT

Returns the width in pixels required to print a given string of text with \texttt{OUTGTEXT} using the current font.

Prototype

\begin{verbatim}
INTERFACE
    FUNCTION GETGTEXTEXTENT (TEXT)
    INTEGER(2) GETGTEXTEXTENT
    CHARACTER(LEN=*) TEXT
    END FUNCTION
END INTERFACE
\end{verbatim}

\texttt{TEXT} Input. \texttt{CHARACTER (LEN=*)}. Text to be analyzed.

Description

Returns the width in pixels that would be required to print a given string of text (including any trailing blanks) with \texttt{OUTGTEXT} using the current font. This function is useful for determining the size of text that uses proportionally spaced fonts. You must initialize fonts with \texttt{INITIALIZEFONTS} before calling any font-related function, including \texttt{GETGTEXTEXTENT}.

Output

The result type is \texttt{INTEGER (2)}. The result is the width of \texttt{text} in pixels if successful; otherwise, -1 (for example, if fonts have not been initialized with \texttt{INITIALIZEFONTS}).
OUTGTEXT

In graphics mode, sends a string of text to the screen, including any trailing blanks.

Prototype

```
INTERFACE
  SUBROUTINE OUTGTEXT (TEXT)
    CHARACTER(LEN=*) TEXT
  END SUBROUTINE
END INTERFACE
```

TEXT
Input. CHARACTER (LEN=*). String to be displayed.

Description

In graphics mode, sends a string of text to the screen, including any trailing blanks. Text output begins at the current graphics position, using the current font set with SETFONT and the current color set with SETCOLORRGB or SETCOLOR. No formatting is provided. After it outputs the text, OUTGTEXT updates the current graphics position.

Before you call OUTGTEXT, you must call INITIALIZEFONTS.

Because OUTGTEXT is a graphics function, the color of text is affected by the SETCOLORRGB function, not by SETTEXTCOLORRGB.

INITIALIZEFONTS

Initializes Windows fonts.

Prototype

```
INTERFACE
```

4-116
FUNCTION INITIALIZEFONTS()
    INTEGER(2) INITIALIZEFONTS
END FUNCTION
END INTERFACE

Description
All fonts in Windows become available after a call to INITIALIZEFONTS. Fonts must be initialized with INITIALIZEFONTS before any other font-related library function (such as GETFONTINFO, GETGTEXTEXTENT, SETFONT, OUTGTEXT) can be used.
For each window you open, you must call INITIALIZEFONTS before calling SETFONT. INITIALIZEFONTS needs to be executed after each new child window is opened in order for a subsequent SETFONT call to be successful.

Output
The result type is INTEGER(2). The result is the number of fonts initialized

SETFONT
Find a single font that matches a specified set of characteristics.

Prototype
INTERFACE
    FUNCTION SETFONT(OPTIONS)
        INTEGER(2) SETFONT
        CHARACTER(LEN=*) OPTIONS
    END FUNCTION
END INTERFACE
OPTIONS

Input. CHARACTER (LEN=*) . String describing font characteristics

Description

Finds a single font that matches a specified set of characteristics and makes it the current font used by the OUTGTEXT function. The SETFONT function searches the list of available fonts for a font matching the characteristics specified in OPTIONS. If a font matching the characteristics is found, it becomes the current font. The current font is used in all subsequent calls to the OUTGTEXT function. There can be only one current font.

The OPTIONS argument consists of letter codes, as follows, that describe the desired font. The OPTIONS parameter is not case sensitive or position sensitive.

t 'fontname'

Name of the desired typeface. It can be any installed font.

hy

Character height, where y is the number of pixels.

wx

Select character width, where x is the number of pixels.

f

Select only a fixed-space font (do not use with the p characteristic).

p

Select only a proportional-space font (do not use with the f characteristic).

v

Select only a vector-mapped font (do not use with the r characteristic). In Windows NT, Roman, Modern, and Script are examples of vector-mapped fonts, also called plotter fonts. True Type fonts (for example, Arial, Symbol, and Times New Roman) are not vector-mapped.

r

Select only a raster-mapped (bitmapped) font (do not use with the v characteristic). In Windows NT®, Courier, Helvetica, and Palatino are examples of raster-mapped fonts, also called screen fonts. True Type fonts are not raster-mapped.

e

Select the bold text format. This parameter is ignored if the font does not allow the bold format.
Select the underline text format. This parameter is ignored if the font does not allow underlining.

Select the italic text format. This parameter is ignored if the font does not allow italics.

Select the font that best fits the other parameters specified.

Select font number \(x\), where \(x\) is less than or equal to the value returned by the INITIALIZEFONTS function.

You can specify as many options as you want, except with \(nx\), which should be used alone. If you specify options that are mutually exclusive (such as the pairs \(f/p\) or \(r/v\)), the SETFONT function ignores them. There is no error detection for incompatible parameters used with \(nx\).

If the \(b\) option is specified and at least one font is initialized, SETFONT sets a font and returns 0 to indicate success.

In selecting a font, the SETFONT routine uses the following criteria, rated from highest precedence to lowest:

- Pixel height
- Typeface
- Pixel width
- Fixed or proportional font

You can also specify a pixel width and height for fonts. If you choose a nonexistent value for either and specify the \(b\) option, SETFONT chooses the closest match.

A smaller font size has precedence over a larger size. If you request Arial 12 with best fit, and only Arial 10 and Arial 14 are available, SETFONT selects Arial 10.

If you choose a nonexistent value for pixel height and width, the SETFONT function applies a magnification factor to a vector-mapped font to obtain a suitable font size. This automatic magnification does not apply if you specify the \(x\) option (raster-mapped font), or if you request a specific typeface and do not specify the \(b\) option (best-fit).

If you specify the \(nx\) parameter, SETFONT ignores any other specified options and supplies only the font number corresponding to \(x\).
If a height is given, but not a width, or vice versa, SETFONT computes the missing value to preserve the correct font proportions.

The font functions affect only OUTGTEXT and the current graphics position; no other Fortran Graphics Library output functions are affected by font usage.

For each window you open, you must call INITIALIZEFONTS before calling SETFONT. INITIALIZEFONTS needs to be executed after each new child window is opened in order for a subsequent SETFONT call to be successful.

**Output**

The result type is INTEGER(2). The result is the index number (x as used in the n.x option) of the font if successful; otherwise, -1.

---

**SETGTEXTROTATION**

*Sets the orientation angle of the font text output in degrees.*

**Prototype**

```fortran
INTERFACE
  SUBROUTINE SETGTEXTROTATION(DEGREES)
    INTEGER(4) DEGREES
  END SUBROUTINE
END INTERFACE
```

**DEGREES**

Input. INTEGER(4). Angle of orientation, in tenths of degrees, of the font text output.
Description

Sets the orientation angle of the font text output in degrees. The current orientation is used in calls to OUTGTEXT. The orientation of the font text output is set in tenths of degrees. Horizontal is 0°, and angles increase counterclockwise so that 900 (90°) is straight up, 1800 (180°) is upside down and left, 2700 (270°) is straight down, and so forth. If the user specifies a value greater than 3600 (360°), the subroutine takes a value equal to:

\[ \text{MODULO (user-specified tenths of degrees, 3600)} \]

Although SETGTEXTROTATION accepts arguments in tenths of degrees, only increments of one full degree differ visually from each other on the screen.

---

GETGTEXTROTATION

*Returns the current orientation of the font text output by OUTGTEXT.*

Prototype

```fortran
INTERFACE
  FUNCTION GETGTEXTROTATION()
    INTEGER(4) GETGTEXTROTATION
  END FUNCTION
END INTERFACE
```

Description

The orientation for text output with OUTGTEXT is set with SETGTEXTROTATION.
Output

The result type is INTEGER(4). It is the current orientation of the font text output in tenths of degrees. Horizontal is 0°, and angles increase counterclockwise so that 900 tenths of degrees (90°) is straight up, 1800 tenths of degrees (180°) is upside-down and left, 2700 tenths of degrees (270°) is straight down, and so forth.

GETTEXTCOLORRGB

*Gets the Red-Green-Blue (RGB) value of the current text color.*

Prototype

```fortran
INTERFACE
    FUNCTION GETTEXTCOLORRGB()
        INTEGER(4) GETTEXTCOLORRGB
    END FUNCTION
END INTERFACE
```

Description

Gets the Red-Green-Blue (RGB) value of the current text color (used with OUTTEXT, WRITE and PRINT). In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you retrieve with GETTEXTCOLORRGB, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

GETTEXTCOLORRGB returns the RGB color value of text over the background color (used by text functions such as OUTTEXT, WRITE, and PRINT), set with SETTEXTCOLORRGB. The RGB color value used for
graphics is set and returned with `SETCOLORRGB` and `GETCOLORRGB`. `SETCOLORRGB` controls the color used by the graphics function `OUTGTEXT`, while `SETTEXTCOLORRGB` controls the color used by all other text output functions. The RGB background color value for both text and graphics is set and returned with `SETBKCOLORRGB` and `GETBKCOLORRGB`.

`SETTEXTCOLORRGB` (and the other RGB color selection functions `SETBKCOLORRGB`, and `SETCOLORRGB`) sets the color to a color value chosen from the entire available range. The non-RGB color functions (`SETTEXTCOLOR`, `SETBKCOLOR`, and `SETCOLOR`) use color indexes rather than true color values. If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

**Output**

The result type is `INTEGER(4)`. It is the RGB value of the current text color.

---

**SETTEXTCOLORRGB**

*Sets the current text color to the specified Red-Green-Blue (RGB) value.*

**Prototype**

```fortran
INTERFACE
  FUNCTION SETTEXTCOLORRGB(COLOR)
    INTEGER(4) SETTEXTCOLORRGB, COLOR
  END FUNCTION
END INTERFACE
```

| COLOR | Input. `INTEGER(4)`. RGB color value to set the text color to. Range and result depend on the system's display adapter. |
**Description**

In each RGB color value, each of the three colors, red, green, and blue, is represented by an eight-bit value (2 hex digits). In the value you specify with `SETTEXTCOLORRGB`, red is the rightmost byte, followed by green and blue. The RGB value's internal structure is as follows:

Larger numbers correspond to stronger color intensity with binary 1111111 (hex FF) the maximum for each of the three components. For example, #0000FF yields full-intensity red, #00FF00 full-intensity green, #FF0000 full-intensity blue, and #FFFFFF full-intensity for all three, resulting in bright white.

`SETTEXTCOLORRGB` sets the current text RGB color. The default value is #00FFFFFF, which is full-intensity white. `SETTEXTCOLORRGB` sets the color used by `OUTTEXT`, `WRITE`, and `PRINT`. It does not affect the color of text output with the `OUTGTEXT` font routine. Use `SETCOLORRGB` to change the color of font output.

`SETBKCOLORRGB` sets the RGB color value of the current background for both text and graphics. `SETCOLORRGB` sets the RGB color value of graphics over the background color, used by the graphics functions such as `ARC`, `FLOODFILLRGB`, and `OUTGTEXT`.

`SETTEXTCOLORRGB` (and the other RGB color selection functions `SETBKCOLORRGB` and `SETCOLORRGB`) sets the color to a value chosen from the entire available range. The non-RGB color functions (`SETTEXTCOLOR`, `SETBKCOLOR`, and `SETCOLOR`) use color indexes rather than true color values.

If you use color indexes, you are limited to the colors available in the palette, at most 256. Some display adapters (SVGA and true color) are capable of creating 262,144 (256K) colors or more. To access any available color, you need to specify an explicit RGB value with an RGB color function, rather than a palette index with a non-RGB color function.

**Output**

The result type is `INTEGER(4)`. The result is the previous text RGB color value.
QuickWin Compatible Support

ABOUTBOXQQ

Specifies the message box information that appears when About command from a Help menu is selected.

Prototype

```
INTERFACE
  FUNCTION ABOUTBOXQQ (STR)
    INTEGER (4) ABOUTBOXQQ
    CHARACTER (LEN=*) STR
  END FUNCTION
END INTERFACE
```

STR Input; output. CHARACTER (LEN=*). Null-terminated C string.

Description

Specifies the information displayed in the message box that appears when the About command from a QuickWin application's Help menu is selected. If your program does not call ABOUTBOXQQ, the QuickWin run-time library supplies a default string.

Output

The value of the result is INTEGER (4). It is zero if successful; otherwise, nonzero.
APPENDMENUQQ

Appends a menu item to the end of a menu and registers its callback subroutine.

Prototype

INTERFACE
FUNCTION APPENDMENUQQ (MENUID, FLAGS, TEXT, ROUTINE)
  LOGICAL APPENDMENUQQ
  INTEGER(4) MENUID, FLAGS
  CHARACTER(LEN=*) TEXT
  EXTERNAL ROUTINE
END FUNCTION
END INTERFACE

MENUID
Input. INTEGER(4). Identifies the menu to which the item is appended, starting with 1 as the leftmost menu.

FLAGS
Input. INTEGER(4). Constant indicating the menu state. Flags can be combined with an inclusive OR. The following constants are available:

$MENUGRAYED Disables and grays out the menu item.
$MENUDISABLED Disables but does not gray out the menu item.
$MENUENABLED Enables the menu item.
$MENUSEPARATOR Draws a separator bar.
$MENUCHECKED Puts a check by the menu item.
$MENUUNCHECKED Removes the check by the menu item.
**TEXT**

Input. CHARACTER (LEN=*) . Menu item name. Must be a null-terminated C string, for example, 'WORDS OF TEXT' C.

**ROUTE**

Input. EXTERNAL. Callback subroutine that is called if the menu item is selected. All routines take a single LOGICAL parameter which indicates whether the menu item is checked or not. You can assign the following predefined routines to menus:

- **WINPRINT**  Prints the program.
- **WINSAVE**  Saves the program.
- **WINEXIT**  Terminates the program.
- **WINSELTEXT**  Selects text from the current window.
- **WINSELGRAPH**  Selects graphics from the current window.
- **WINSELALL**  Selects the entire contents of the current window.
- **WINCOPY**  Copies the selected text and/or graphics from the current window to the Clipboard.
- **WINPASTE**  Allows the user to paste Clipboard contents (text only) to the current text window of the active window during a READ.
- **WINCLEARPASTE**  Clears the paste buffer.
- **WINSIZETOFIT**  Sizes output to fit window.
- **WINFULLSCREEN**  Displays output in full screen.
- **WINSTATE**  Toggles between pause and resume states of text output.
- **WINCASCADE**  Cascades active windows.
- **WINTILE**  Tiles active windows.
- **WINARRANGE**  Arranges icons.
WINSTATUS Enables a status bar.
WININDEX Displays the index for QuickWin help.
WINUSING Displays information on how to use Help.
WINABOUT Displays information about the current QuickWin application.
NUL No callback routine.

Description
You do not need to specify a menu item number, because APPENDMENUQQ always adds the new item to the bottom of the menu list. If there is no item yet for a menu, your appended item is treated as the top-level menu item (shown on the menu bar), and TEXT becomes the menu title. APPENDMENUQQ ignores the callback routine for a top-level menu item if there are any other menu items in the menu. In this case, you can set ROUTINE to NUL.

If you want to insert a menu item into a menu rather than append to the bottom of the menu list, use INSERTMENUQQ.

The constants available for flags can be combined with an inclusive OR where reasonable, for example $MENUCHECKED .OR. $MENUNABLED. Some combinations do not make sense, such as $MENUNABLED and $MENUEABLED, and lead to undefined behavior.

You can create quick-access keys in the text strings you pass to APPENDMENUQQ as TEXT by placing an ampersand (&) before the letter you want underlined. For example, to add a Print menu item with the R underlined, TEXT should be "P&rint". Quick-access keys allow users of your program to activate that menu item with the key combination ALT+QUICK-ACCESS-KEY (ALT+R in the example) as an alternative to selecting the item with the mouse.

Output
The result type is LOGICAL: .TRUE. if successful; otherwise, .FALSE..
CLICKMENUQQ

Simulates the effect of clicking or selecting a menu command.

Prototype

INTERFACE
   FUNCTION CLICKMENUQQ(ITEM)
       INTEGER(4) CLICKMENUQQ, ITEM
   END FUNCTION
END INTERFACE

ITEM
Input. INTEGER(4). Constant that represents the command selected from the Window menu. Must be one of the following symbolic constants:

- QWIN$STATUS Status command
- QWIN$TILE Tile command
- QWIN$CASCADE Cascade command
- QWIN$ARRANGE Arrange Icons command

Description
Simulates the effect of clicking or selecting a menu command. The QuickWin application responds as though the user had clicked or selected the command.

Output
The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero.
DELETEMENUQQ

Deletes a menu item from a QuickWin menu.

Prototype

```fortran
INTERFACE
  FUNCTION DELETEMENUQQ(MENUID, ITEMID)
    LOGICAL DELETEMENUQQ
    INTEGER(4) MENUID, ITEMID
  END FUNCTION
END INTERFACE
```

MENUID
Input. INTEGER(4). Identifies the menu that contains the menu item to be deleted, starting with 1 as the leftmost menu.

ITEMID
Input. INTEGER(4). Identifies the menu item to be deleted, starting with 0 as the top menu item.

Description

Deletes a menu item from a QuickWin menu.

Output

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
FOCUSQQ

Sets focus to the window with the specified unit number.

Prototype

IA-32

interface FOCUSQQ

FUNCTION FOCUSQQI4(IUNIT)

!MS$ ATTRIBUTES ALIAS:'_ILF_FOCUSQQI4'::FOCUSQQI4
  integer(4) FOCUSQQI4, IUNIT
END FUNCTION

end interface

Itanium®-based systems

interface FOCUSQQ

FUNCTION FOCUSQQI4(IUNIT)

!MS$ ATTRIBUTES ALIAS:'_ILF_FOCUSQQI4'::FOCUSQQI4
  integer(4) FOCUSQQI4, IUNIT
END FUNCTION

FUNCTION FOCUSQQI8(IUNIT)

!MS$ ATTRIBUTES ALIAS:'_ILF_FOCUSQQ'::FOCUSQQI8
  integer(4) FOCUSQQI8
  integer(8) IUNIT
END FUNCTION

end interface

IUNIT

Input: for IA-32, integer(4) IUNIT; for Itanium-based systems, there two entry points: integer(4) IUNIT and integer(8) IUNIT. Unit number of the window to which the focus is set. Unit numbers 0, 5, and 6 refer to the default startup window.
Description

Units 0, 5, and 6 refer to the default window only if the program does not specifically open them. If these units have been opened and connected to windows, they are automatically reconnected to the console once they are closed.

Unlike SETACTIVEQQ, FOCUSQQ brings the specified unit to the foreground. Note that the window with the focus is not necessarily the active window (the one that receives graphical output). A window can be made active without getting the focus by calling SETACTIVEQQ.

A window has focus when it is given the focus by FOCUSQQ, when it is selected by a mouse click, or when an I/O operation other than a graphics operation is performed on it, unless the window was opened with IOFOCUS=.FALSE.. The IOFOCUS specifier determines whether a window receives focus when an I/O statement is executed on that unit. For example:

```
OPEN (UNIT = 10, FILE = 'USER', IOFOCUS = .TRUE.)
```

By default IOFOCUS=.TRUE., except for child windows opened with as unit *. If IOFOCUS=.TRUE., the child window receives focus prior to each READ, WRITE, PRINT, or OUTTEXT. Calls to graphics functions (such as OUTGTEXT and ARC) do not cause the focus to shift.

Output

The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero.

---

**GETACTIVEQQ**

*Returns the unit number of the currently active child window.*

**Prototype**

IA-32
INTERFACE
FUNCTION GETACTIVEQQ()
  INTEGER(4) GETACTIVEQQ
  !MS$ ATTRIBUTES C, ALIAS:'_wggetactiveunit':: 
  &
  GETACTIVEQQ
END FUNCTION
END INTERFACE

Itanium®-based systems
INTERFACE
FUNCTION GETACTIVEQQ()
  INTEGER(8) GETACTIVEQQ
  !MS$ ATTRIBUTES C, ALIAS:'_wggetactiveunit':: &
  GETACTIVEQQ
END FUNCTION
END INTERFACE

Description
This function returns the unit number of the currently active child window.

Output
The result type is INTEGER(4) (IA-32) or INTEGER(8) (Itanium-based systems). The result is the unit number of the currently active window. Returns the parameter QWIN$NOACTIVEWINDOW if no child window is active.

GETEXITQQ
Gets the setting for a QuickWin application's exit behavior.

Prototype
INTERFACE
FUNCTION GETEXITQQ()
  INTEGER(4) GETEXITQQ
END FUNCTION
END INTERFACE

Description
This function gets the setting for a QuickWin application's exit behavior.

Output
The result type is INTEGER(4). The result is exit mode with one of the following constants:

**QWIN$EXITPROMPT**
Displays a message box that reads "Program exited with exit status \( n \). Exit Window?", where \( n \) is the exit status from the program. If the user chooses Yes, the application closes the window and terminates. If the user chooses No, the dialog box disappears and the user can manipulate the window as usual. The user must then close the window manually.

**QWIN$EXITNOPERSIST**
Terminates the application without displaying a message box.

**QWIN$EXITPERSIST**
Leaves the application open without displaying a message box.

The default for both QuickWin and Console Graphics applications is QWIN$EXITPROMPT.
GETHWndQQ

Converts a window unit number into a Windows handle.

Prototype

IA-32

INTERFACE
  FUNCTION GETHWndQQ(IUNIT)
    INTEGER(4) GETHWndQQ, IUNIT
  END FUNCTION
END INTERFACE

Itanium-based systems

INTERFACE
  FUNCTION GETHWndQQ(IUNIT)
    INTEGER(8) GETHWndQQ, IUNIT
  END FUNCTION
END INTERFACE

IUNIT  Input. INTEGER(4) (IA-32) or INTEGER(8) (Itanium-based systems). Window unit number. If IUNIT is set to QWIN$FRAMEWINDOW, the handle of the frame window is returned.

Description

This function converts a window unit number into a Windows handle.

Output

The result type is INTEGER(4) (IA-32) or INTEGER(8) (Itanium-based systems). The result is a true Windows handle to the window. It returns -1 if IUNIT is not open.
GETUNITQQ

Returns the unit number corresponding to the specified Windows handle.

Prototype

IA-32

INTERFACE
  FUNCTION GETUNITQQ(IHANDLE)
    INTEGER(4) GETUNITQQ,IHANDLE
  END FUNCTION
END INTERFACE

Itanium-based systems

INTERFACE
  FUNCTION GETUNITQQ(IHANDLE)
    INTEGER(4) GETUNITQQ,IHANDLE
  END FUNCTION
END INTERFACE

IHANDLE  Input. INTEGER(4) (IA-32) or INTEGER(8) (Itanium-based systems). The Windows handle to the window; this is a unique ID.

Description

Returns the unit number corresponding to the specified Windows handle.
This routine is the inverse of GETHWNDQQ.

Output

The result type is INTEGER(4) (IA-32) or INTEGER(8) (Itanium-based systems). The result is the unit number corresponding to the specified Windows handle. It returns -1 if IHANDLE does not exist.
GETWINDOWCONFIG

*Gets the properties of the current window.*

**Prototype**

```plaintext
INTERFACE
  FUNCTION GETWINDOWCONFIG(WC)
    LOGICAL GETWINDOWCONFIG
    STRUCTURE /WINDOWCONFIG/
    INTEGER(2) NUMXPIXELS, NUMYPIXELS
    INTEGER(2) NUMTEXTCOLS, NUMTEXTROWS
    INTEGER(2) NUMCOLORS
    INTEGER(2) FONTSIZE
    CHARACTER(LEN=80) TITLE
    INTEGER(2) BITSPERPIXEL
    INTEGER(2) NUMVIDEOPAGES
    INTEGER(2) MODE
    INTEGER(2) ADAPTER
    INTEGER(2) MONITOR
    INTEGER(2) MEMORY
    INTEGER(2) ENVIRONMENT
    CHARACTER(LEN=32) EXTENDFONTNAME
    INTEGER(4) EXTENDFONTSIZE
    INTEGER(4) EXTENDFONTATTIBUTES
    END STRUCTURE
    RECORD /WINDOWCONFIG/ WC
    END FUNCTION
  END INTERFACE

WC Output. RECORD /WINDOWCONFIG/. Contains window properties.

STRUCTURE /WINDOWCONFIG/
```
INTEGER(2) NUMXPIXELS ! Number of pixels on x-axis
INTEGER(2) NUMYPIXELS ! Number of pixels on y-axis
INTEGER(2) NUMTEXTCOLS ! Number of text columns
! available
INTEGER(2) NUMTEXTROWS ! Number of text rows
! available
INTEGER(2) NUMCOLORS ! Number of color indexes
INTEGER(4) FONTSIZE ! Size of default font. Set
! to QWIN$EXTENDFONT when using
! multibyte characters, in which case
! EXTENFONTSIZE sets the font size.
CHARACTER(LEN=80) title ! window title
INTEGER(2) BITSPERPIXEL ! number of bits per pixel
! The next three parameters support multibyte
! character sets (such as Japanese)
CHARACTER(LEN=32) EXTENDFONTNAME ! any
! nonproportionally spaced font available on the
! system
INTEGER(4) EXTENFONTSIZE ! takes same values as
! FONTSIZE, but used for multibyte
! character sets when FONTSIZE set to
! QWIN$EXTENDFONT
INTEGER(4) EXTENDFONTATTRIBUTES ! font attributes
! such as bold and italic for
! multibyte character sets
END STRUCTURE

Description

GETWINDOWCONFIG returns information about the active child window. If you have not set the window properties with SETWINDOWCONFIG,
GETWINDOWCONFIG returns default window values.

A typical set of values would be 1024 x pixels, 768 y pixels, 128 text
columns, 48 text rows, and a font size of 8x16 pixels. The resolution of the
display and the assumed font size of 8x16 pixels generates the number of
text rows and text columns. The resolution (in this case, 1024 x pixels by
768 \( Y \) pixels) is the size of the virtual window. To get the size of the physical window visible on the screen, use \texttt{GETWSIZEQQ}. In this case, \texttt{GETWSIZEQQ} returned the following values: (0,0) for the x and y position of the physical window, 25 for the height or number of rows, and 71 for the width or number of columns.

The number of colors returned depends on the video drive. The window title defaults to "Graphic1" for the default window. All of these values can be changed with \texttt{SETWINDOWCONFIG}.

Note that the \texttt{bitsperpixel} field in the \texttt{STRUCTURE /WINDOWCONFIG/} is an output field only, while the other fields return output values to \texttt{GETWINDOWCONFIG} and accept input values from \texttt{SETWINDOWCONFIG}.

\textbf{Output}

The result type is \texttt{LOGICAL(4)}. The result is \texttt{.TRUE.} if successful; otherwise, \texttt{.FALSE.} (for example, if there is no active child window).

---

\textbf{GETWSIZEQQ}

\textit{Gets the size and position of a window.}

\textbf{Prototype}

\texttt{INTERFACE}

\texttt{FUNCTION GETWSIZEQQ(IUNIT,IREQ,WINFO)\
  STRUCTURE /QWINFO/\
    INTEGER(2) TYPE,X,Y,H,W\
  END STRUCTURE\
  INTEGER(4) GETWSIZEQQ,IUNIT\
  INTEGER(4) IREQ\
  RECORD /QWINFO/ WINFO\
  END FUNCTION\
END INTERFACE}
UNIT

Input. INTEGER(4). Specifies the window unit. Unit numbers 0, 5 and 6 refer to the default startup window only if you have not explicitly opened them with the OPEN statement. To access information about the frame window (as opposed to a child window), set unit to the symbolic constant QWIN$FRAMEWINDOW.

IREQ

Input. INTEGER(4). Specifies what information is obtained.

\begin{itemize}
  \item QWIN$SIZEMAX Gets information about the maximum window size.
  \item QWIN$SIZECURR Gets information about the current window size.
\end{itemize}

INFO

Output. RECORD /QINFO/. Physical coordinates of the window's upper-left corner, and the current or maximum height and width of the window's client area (the area within the frame).

\begin{verbatim}
STRUCTURE /QINFO/
  INTEGER(2) TYPE ! request type (controls
                  ! SETWSIZEQQ)
  INTEGER(2) X ! x coordinate for upper left
  INTEGER(2) Y ! y coordinate for upper left
  INTEGER(2) H ! window height
  INTEGER(2) W ! window width
END STRUCTURE
\end{verbatim}

Description

The position and dimensions of child windows are expressed in units of character height and width. The position and dimensions of the frame window are expressed in screen pixels.

The height and width returned for a frame window reflects the size in pixels of the client area excluding any borders, menus, and status bar at the bottom of the frame window. You should adjust the values used in SETWSIZEQQ to take this into account.

The client area is the area actually available to place child windows.
Output

The result type is \texttt{INTEGER(4)}. The result is zero if successful; otherwise, nonzero.

\textbf{INQFOCUSQQ}

Determines which window has the focus.

\textbf{Prototype}

\textbf{IA-32}

\begin{verbatim}
INTERFACE
  FUNCTION INQFOCUSQQI4(IUNIT)
    !MS$ ATTRIBUTES ALIAS:'_Ilf_INQFOCUSQQI4':: &
    INQFOCUSQQI4
      INTEGER(4) INQFOCUSQQI4,IUNIT
    END FUNCTION
END INTERFACE
\end{verbatim}

\textbf{Itanium-based systems}

\begin{verbatim}
interface INQFOCUSQQ
  FUNCTION INQFOCUSQQI4(IUNIT)
    !MS$ ATTRIBUTES ALIAS:'_Ilf_INQFOCUSQQI4':: &
    INQFOCUSQQI4
    END FUNCTION

  FUNCTION INQFOCUSQQI8(IUNIT)
    !MS$ ATTRIBUTES ALIAS:'_Ilf_INQFOCUSQQ':: &
    INQFOCUSQQI8
    INTEGER(4) INQFOCUSQQI8
    INTERGER(8) IUNIT
    END FUNCTION
END INTERFACE
\end{verbatim}
UNIT Output: for IA-32, integer(4) IUNIT; for Itanium-based systems, there are two entry points: integer(4) IUNIT and integer(8) IUNIT. Unit number of the window that has the I/O focus.

Description

Unit numbers 0, 5, and 6 refer to the default window only if the program has not specifically opened them. If these units have been opened and connected to windows, they are automatically reconnected to the console once they are closed.

The window with focus is always in the foreground. Note that the window with the focus is not necessarily the active window (the one that receives graphical output). A window can be made active without getting the focus by calling SETACTIVEQQ.

A window has focus when it is given the focus by FOCUSQQ, when it is selected by a mouse click, or when an I/O operation other than a graphics operation is performed on it, unless the window was opened with IOFOCUS=.FALSE.. The IOFOCUS specifier determines whether a window receives focus when an I/O statement is executed on that unit. For example:

OPEN (UNIT = 10, FILE = 'USER', IOFOCUS = .TRUE.)

By default IOFOCUS=.TRUE., except for child windows opened with as unit *. If IOFOCUS=.TRUE., the child window receives focus prior to each READ, WRITE, PRINT, or OUTTEXT. Calls to graphics functions (such as OUTGTEXT and ARC) do not cause the focus to shift.

Output

The result type is INTEGER(4). The result is zero if successful; otherwise, nonzero. The function fails if the window with the focus is associated with a closed unit.
**INSERTMENUQQ**

*Inserts a menu item into a QuickWin menu and registers its callback routine.*

**Prototype**

```plaintext
INTERFACE
FUNCTION
   INSERTMENUQQ(MENUID,ITEMID,FLGS,TEXT,ROUTINE)
   LOGICAL INSERTMENUQQ
   INTEGER(4) MENUID,ITEMID,FLAGS
   CHARACTER(LEN=*) TEXT
   EXTERNAL ROUTINE
END FUNCTION
END INTERFACE
```

**MENUID**

Input. INTEGER(4). Identifies the menu in which the item is inserted, starting with 1 as the leftmost menu.

**ITEMID**

Input. INTEGER(4). Identifies the position in the menu where the item is inserted, starting with 0 as the top menu item.

**FLAGS**

Input. INTEGER(4). Constant indicating the menu state. Flags can be combined with an inclusive OR (see Results section below). The following constants are available:

- **$MENUGRAYED**
  - Disables and grays out the menu item.
- **$MENUDISABLED**
  - Disables but does not gray out the menu item.
- **$MENUENABLED**
  - Enables the menu item.
- **$MENUSEPARATOR**
  - Draws a separator bar.
- **$MENUCHECKED**
  - Puts a check by the menu item.
$MENUUNCHECKED  Removes the check by the menu item.

TEXT  Input. CHARACTER (LEN=*) . Menu item name. Must be a null-terminated C string, for example, words of text 'C.'

ROUTINE  Input. EXTERNAL. Callback subroutine that is called if the menu item is selected. All routines must take a single LOGICAL parameter which indicates whether the menu item is checked or not. You can assign the following predefined routines to menus:

WINPRINT  Prints the program.
WINSAVE  Saves the program.
WINEXIT  Terminates the program.
WINSELTEXT  Selects text from the current window.
WINSELGRAPH  Selects graphics from the current window.
WINSELALL  Selects the entire contents of the current window.
WCOPY  Copies the selected text and/or graphics from current window to the Clipboard.
WINPASTE  Allows the user to paste Clipboard contents (text only) to the current text window of the active window during a READ.
WINCLEARPASTE  Clears the paste buffer.
WINSIZEToFit  Sizes output to fit window.
WINFULLSCREEN  Displays output in full screen.
WINSTATE  Toggles between pause and resume states of text output.
WINCASCADE  Cascades active windows.
WINTILE  Tiles active windows.
WINARRANGE
Arranges icons.

WINSTATUS
Enables a status bar.

WININDEX
Displays the index for QuickWin help.

WINUSING
Displays information on how to use Help.

WINABOUT
Displays information about the current QuickWin application.

NUL
No callback routine.

Description
Menus and menu items must be defined in order from left to right and top to bottom. For example, INSERTMENUQQ fails if you try to insert menu item 7 when 5 and 6 are not defined yet. For a top-level menu item, the callback routine is ignored if there are subitems under it.

The constants available for flags can be combined with an inclusive OR where reasonable, for example $MENUCHECKED .OR. $MENUENABLED. Some combinations do not make sense, such as $MENUENABLED and $MENUDISABLED, and lead to undefined behavior.

You can create quick-access keys in the text strings you pass to INSERTMENUQQ as text by placing an ampersand (&) before the letter you want underlined. For example, to add a Print menu item with the r underlined, text should be "P&rint". Quick-access keys allow users of your program to activate that menu item with the key combination ALT+QUICK-ACCESS-KEY (ALT+R in the example) as an alternative to selecting the item with the mouse.

Output
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
MESSAGEBOXQQ

Displays a message box in a QuickWin window.

Prototype

INTERFACE
  FUNCTION MESSAGEBOXQQ(MSG, CAPTION, MTYPE)
    CHARACTER(LEN=*) MSG, CAPTION
    INTEGER(4) MESSAGEBOXQQ, MTYPE
  END FUNCTION
END INTERFACE

MSG
  Input. CHARACTER(LEN=*) . Null-terminated C string.
  Message the box displays.

CAPTION
  Input. CHARACTER(LEN=*) . Null-terminated C string.
  Caption that appears in the title bar.

MTYPE
  Input. INTEGER(4). Symbolic constant that determines
  the objects (buttons and icons) and attributes of the
  message box. You can combine several constants using
  an inclusive OR (IOR or OR). The symbolic constants
  and their associated objects or attributes are:

  MB$ABORTRETRYIGNORE The Abort, Retry, and Ignore buttons.

  MB$DEFBUTTON1 The first button is the default.

  MB$DEFBUTTON2 The second button is the default.

  MB$DEFBUTTON3 The third button is the default.

  MB$ICONASTERISK Lowercase i in blue circle icon.

  MB$ICONEXCLAMATION The exclamation-mark icon.
MB$ICONHAND The stop-sign icon.
MB$ICONINFORMATION Lowercase / in blue circle icon.
MB$ICONQUESTION The question-mark icon.
MB$ICONSTOP The stop-sign icon.
MB$OK The OK button.
MB$OKCANCEL The OK and Cancel buttons.
MB$RETRYCANCEL The Retry and Cancel buttons.
MB$SYSTEMMODAL Box is system-modal: all applications are suspended until the user responds.
MB$YESNO The Yes and No buttons.
MB$YESNOCANCEL The Yes, No, and Cancel buttons.

**Description**
This function displays a message box in a QuickWin window.

**Output**
The result type is INTEGER(4). The result is zero if memory is not sufficient for displaying the message box. Otherwise, the result is one of the following values, indicating the user's response to the message box:

MB$IDABORT The Abort button was pressed.
MB$IDCANCEL The Cancel button was pressed.
MB$IDIGNORE The Ignore button was pressed.
MB$IDNO The No button was pressed.
MB$IDOK The OK button was pressed.
MB$IDRETRY The Retry button was pressed.
MB$IDYES The Yes button was pressed.
**MODIFYMENUFLAGSQQ**

*Modifies a menu item’s state.*

**Prototype**

```fortran
INTERFACE
FUNCTION MODIFYMENUFLAGSQQ(MENUID, ITEMID, FLAGS)
  LOGICAL MODIFYMENUFLAGSQQ
  INTEGER(4) MENUID, ITEMID, FLAGS
END FUNCTION
END INTERFACE
```

**MENUID**

Input. INTEGER (4). Identifies the menu containing the item whose state is to be modified, starting with 1 as the leftmost menu.

**ITEMID**

Input. INTEGER (4). Identifies the menu item whose state is to be modified, starting with 0 as the top item.

**FLAGS**

Input. INTEGER (4). Constant indicating the menu state. Flags can be combined with an inclusive OR. The following constants are available:

- `$MENUGRAYED` Disables and grays out the menu item.
- `$MENUDISABLED` Disables but does not gray out the menu item.
- `$MENUENABLED` Enables the menu item.
- `$MENUSEPARATOR` Draws a separator bar.
- `$MENUCHECKED` Puts a check by the menu item.
- `$MENUUNCHECKED` Removes the check by the menu item.
**Description**

The constants available for flags can be combined with an inclusive OR where reasonable, for example $MENUCHECKED .OR. $MENUNABLERED. Some combinations do not make sense, such as $MENUNABLERED and $MENUDISABLED, and lead to undefined behavior.

**Output**

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

---

**MODIFYMENUROUTINEQQ**

*Changes a menu item's callback routine.*

**Prototype**

```
INTERFACE
  FUNCTION MODIFYMENUROUTINEQQ(MENUID, ITEMID, ROUTINE)
    LOGICAL MODIFYMENUROUTINEQQ
    INTEGR(4) MENUID, ITEMID
    EXTERNAL ROUTINE
  END FUNCTION
END INTERFACE
```

- **MENUID**
  - Input. INTEGER (4). Identifies the menu that contains the item whose callback routine is to be changed, starting with 1 as the leftmost menu.

- **ITEMID**
  - Input. INTEGER (4). Identifies the menu item whose callback routine is to be changed, starting with 0 as the top item.
ROUTINE Input. EXTERNAL. Callback subroutine called if the menu item is selected. All routines must take a single LOGICAL parameter that indicates whether the menu item is checked or not. The following predefined routines are available for assigning to menus:

WINPRINT Prints the program.
WINSAVE Saves the program.
WINEXIT Terminates the program.
WINSELTEXT Selects text from the current window.
WINSELGRAPH Selects graphics from the current window.
WINSELALL Selects the entire contents of the current window.
WINCOPY Copies the selected text and/or graphics from the current window to the Clipboard.
WINPASTE Allows the user to paste Clipboard contents (text only) to the current text window of the active window during a READ.
WINCLEARPASTE Clears the paste buffer.
WINSIZETOFIT Sizes output to fit window.
WINFULLSCREEN Displays output in full screen.
WINSTATE Toggles between pause and resume states of text output.
WINCASCADE Cascades active windows.
WINTILE Tiles active windows.
WINARRANGE Arranges icons.
WINSTATUS Enables a status bar.
WININDEX Displays the index for QuickWin Help.
WINUSING Displays information on Help.
WINABOUT Displays information about the current QuickWin application.
NUL No callback routine.

Description
This function changes a menu item's callback routine.

Output
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..

MODIFYMENUSTRINGQQ
Changes a menu item's text string.

Prototype
INTERFACE
FUNCTION MODIFYMENUSTRINGQQ(MENUID,ITEMID,TEXT)
   LOGICAL MODIFYMENUSTRINGQQ
   INTEGER(4) MENUID,ITEMID
   CHARACTER(LEN=*) TEXT
END FUNCTION
END INTERFACE

MENUID Input. INTEGER(4). Identifies the menu containing the item whose text string is to be changed, starting with 1 as the leftmost item.
ITEMID Input. INTEGER(4). Identifies the menu item whose text string is to be changed, starting with 0 as the top menu item.
TEXT Input. CHARACTER (LEN=*) . Menu item name. Must be a null-terminated C string. For example, words of text 'C.'

Description
You can add access keys in your text strings by placing an ampersand (&) before the letter you want underlined. For example, to add a Print menu item with the r underlined, use "Print" as TEXT.

Output
The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
Itanium-based systems

INTERFACE

FUNCTION REGISTERMOUSEEVENT (UNIT, MouseEvents, &

   CallBackRoutine)

   INTEGER(4) REGISTERMOUSEEVENT
   !MS$ ATTRIBUTES C, ALIAS:'registermouseeventqq' &
   :: registermouseevent
   INTEGER(8) UNIT
   INTEGER(4) MouseEvents
   EXTERNAL CallBackRoutine

END FUNCTION
END INTERFACE

UNIT

   Input. INTEGER(4) (IA-32) or INTEGER(8)
   (Itanium-based systems). Unit number of the window
   whose callback routine on mouse events is to be
   registered.

MOUSEEVENTS

   Input. INTEGER(4). One or more mouse events to be
   handled by the callback routine to be registered.
   Symbolic constants for the possible mouse events are:
   
   MOUSE$LBUTTONDOWN
       Left mouse button down
   MOUSE$LBUTTONUP
       Left mouse button up
   MOUSE$LBUTTONDBLCLK
       Left mouse button double-click
   MOUSE$RBUTTONDOWN
       Right mouse button down
   MOUSE$RBUTTONUP
       Right mouse button up
   MOUSE$RBUTTONDBLCLK
       Right mouse button double-click
   MOUSE$MOVE
       Mouse moved
CALLBACKROUTINE
Input. EXTERNAL. Routine to be called on specified mouse event in the specified window.

Description
Registers the application-supplied callback routine to be called when a specified mouse event occurs in a specified window. For every BUTTONDOWN or BUTTONDBLCLK event there is an associated BUTTONUP event. When the user double clicks, four events happen: BUTTONDOWN and BUTTONUP for the first click, and BUTTONDBLCLK and BUTTONUP for the second click. The difference between getting BUTTONDBLCLK and BUTTONDOWN for the second click depends on whether the second click occurs in the double click interval, set in the system's CONTROL PANEL/MOUSE.

Output
The result type is INTEGER(4). The result is zero or a positive integer if successful; otherwise, a negative integer that can be one of the following:

MOUSE$BADUNIT The unit specified is not open, or is not associated with a QuickWin window.

MOUSE$BADEVENT The event specified is not supported.

SETACTIVEQQ
Makes a child window active, but does not give it focus.

Prototype
IA-32
INTERFACE
FUNCTION SETACTIVEQQ(UNIT)
   INTEGER(4) SETACTIVEQQ, UNIT
   !MS$ ATTRIBUTES C, ALIAS:'_wgsetactiveunit':: &
   SETACTIVEQQ
END FUNCTION
END INTERFACE

**Itanium®-based systems**

INTERFACE
   FUNCTION SETACTIVEQQ(UNIT)
      INTEGER(8) SETACTIVEQQ, UNIT
      !MS$ ATTRIBUTES C, ALIAS:'_wgsetactiveunit':: &
      SETACTIVEQQ
   END FUNCTION
END INTERFACE

UNIT Input. INTEGER(4) (IA-32) or INTEGER(8) (Itanium-based systems). Unit number of the child window to be made active.

**Description**

When a window is made active, it receives graphics output (from ARC, LINETO and OUTGTEXT, for example) but is not brought to the foreground and does not have the focus. If a window needs to be brought to the foreground, it must be given the focus. A window is given focus with FOCUSQQ, by clicking it with the mouse, or by performing I/O other than graphics on it, unless the window was opened with IFOCUS='.'FALSE.' By default, IFOCUS='.'TRUE.', except for child windows opened as unit '.*'.

The window that has the focus is always on top, and all other windows have their title bars grayed out. A window can have the focus and yet not be active and not have graphics output directed to it. Graphical output is independent of focus.

If IFOCUS='.'TRUE.', the child window receives focus prior to each READ, WRITE, PRINT, or OUTTEXT. Calls to graphics functions (such as OUTGTEXT and ARC) do not cause the focus to shift.
Output
The result type is INTEGER(4) (IA-32) or INTEGER(8) (Itanium-based systems). The result is 1 if successful; otherwise, 0.

SETEXITQQ
Sets a QuickWin application's exit behavior.

Prototype

```fortran
INTERFACE
  FUNCTION SETEXITQQ(EXITMODE)
    INTEGER(4) SETEXITQQ,EXITMODE
  END FUNCTION
END INTERFACE
```

EXITMODE 
Input. INTEGER(4). Determines the program exit behavior. The following exit parameters:

- QWIN$EXITPROMPT
  Displays the following message box:
  "Program exited with exit status X. Exit Window?" where X is the exit status from the program. If Yes is entered, the application closes the window and terminates. If No is entered, the dialog box disappears and you can manipulate the windows as usual. You must then close the window manually.

- QWIN$EXITNOPERSIST
  Terminates the application without displaying a message box.

- QWIN$EXITPERSIST
  Leaves the application open without displaying a message box.
**Description**
This function sets a QuickWin application's exit behavior.

**Output**
The result type is INTEGER (4). The result is zero if successful; otherwise, a negative value. The default for both QuickWin and Standard Graphics applications is QWIN$EXITPROMPT.

---

**SETMESSAGEQQ**
*Changes QuickWin status messages, state messages, and dialog box messages.*

---

**Prototype**

```fortran
INTERFACE
SUBROUTINE SETMESSAGEQQ(MSG, ID)
  CHARACTER(LEN=*) MSG
  INTEGER(4) ID
END SUBROUTINE
END INTERFACE
```

**MSG**
Input. CHARACTER (LEN=*) . Message to be displayed. Must be a regular Fortran string, not a C string. Can include multibyte characters.

**ID**
Input. INTEGER (4) . Identifier of the message to be changed. The following table shows the messages that can be changed and their identifiers:

<table>
<thead>
<tr>
<th>ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>QWIN$MSG_TERM</td>
<td>&quot;Program terminated with exit code&quot;</td>
</tr>
<tr>
<td>QWIN$MSG_EXITQ</td>
<td>&quot;\nExit Window?&quot;</td>
</tr>
</tbody>
</table>
Description

You can change any string produced by QuickWin by calling \texttt{SETMESSAGEQQ} with the appropriate \texttt{ID}. This includes status messages displayed at the bottom of a QuickWin application, state messages (such as "Paused"), and dialog box messages. These messages can include multibyte characters.
SETWINDOWCONFIG

Sets the properties of a child window.

Prototype

INTERFACE
  FUNCTION SETWINDOWCONFIG(wc)
    logical SETWINDOWCONFIG
  STRUCTURE /WINDOWCONFIG/
    INTEGER(2) NUMXPIXELS, NUMYPIXELS
    INTEGER(2) NUMTEXTCOLUMNS, NUMTEXTROWS
    INTEGER(2) NUMCOLORS
    INTEGER(4) FONTSIZE
    CHARACTER(LEN=80) TITLE
    INTEGER(2) BITSPERPIXEL
    INTEGER(2) NUMVIDEOPAGES
    INTEGER(2) MODE
    INTEGER(2) ADAPTER
    INTEGER(2) MONITOR
    INTEGER(2) MEMORY
    INTEGER(2) ENVIRONMENT
    CHARACTER(LEN=32) EXTENDFONTCOMPATIBILITY
    INTEGER(4) EXTENDFONTSIZE
    INTEGER(4) EXTENDFONTSIZE
    INTEGER(4) EXTENDFONTATTRIBUTES
  END STRUCTURE
  RECORD /WINDOWCONFIG/WC
END FUNCTION /WINDOWCONFIG/WC
END INTERFACE

WC

Input: RECORD /WINDOWCONFIG/. Contains window properties.

STRUCTURE /WINDOWCONFIG/

INTEGER(2) NUMXPIXELS ! Number of pixels on x-axis
INTEGER(2) NUMYPXELS ! Number of pixels on y-axis
INTEGER(2) NUMTEXTCOLS ! Number of text columns
           ! available
INTEGER(2) NUMTEXTROWS ! Number of text rows
           ! available
INTEGER(2) NUMCOLORS ! Number of color indexes
INTEGER(4) FONTSIZE ! Size of default font. Set
           ! to QWIN$EXTENDFONTSIZE when using
           ! multibyte characters, in which case
           ! EXTENDFONTSIZE sets the font size.
CHARACTER(LEN=80) TITLE ! window title, a C string
           ! The next three parameters support multibyte
           ! character sets (such as Japanese)
CHARACTER(LEN=32) EXTENDFONTNAME ! any
           ! nonproportionally spaced font available on the
           ! system
INTEGER(4) EXTENDFONTSIZE ! takes same values as
           ! FONTSIZE, but used for multibyte
           ! character sets when FONTSIZE set to
           ! QWIN$EXTENDFONTSIZE
INTEGER(4) EXTENDFONTATTRIBUTES ! font attributes
           ! such as bold and italic for
           ! multibyte character sets
END STRUCTURE

Description

If you use SETWINDOWCONFIG to set the variables in WINDOWCONFIG to
-1, the function sets the highest resolution possible for your system, given
the other fields you specify, if any. You can set the actual size of the window
by specifying parameters that influence the window size: the number of x
and y pixels, the number of rows and columns, and the font size. If you do
not call SETWINDOWCONFIG, the window defaults to the best possible
resolution and a font size of 8x16. The number of colors available depends
on the video driver used.
If you use SETWINDOWCONFIG, you should specify a value for each field (-1 or your own value for the numeric fields and a C string for the title, for example, "words of text"C). Using SETWINDOWCONFIG with only some fields specified can result in useless values for the unspecified fields.

If you request a configuration that cannot be set, SETWINDOWCONFIG returns .FALSE. and calculates parameter values that will work and are as close as possible to the requested configuration. A second call to SETWINDOWCONFIG establishes the adjusted values; for example:

```plaintext
status = SETWINDOWCONFIG(WC)
if (.NOT.status) status = SETWINDOWCONFIG(WC)
```

If you specify values for all four of the size parameters, NUMXPIXELS, NUMYPixels, NUMTEXTCOLS, and NUMTEXTROWS, the font size is calculated by dividing these values. The default font is Courier New and the default font size is 8x16. There is no restriction on font size, except that the window must be large enough to hold it.

Under Standard Graphics, the application attempts to start in Full Screen mode with no window decoration (window decoration includes scroll bars, menu bar, title bar, and message bar) so that the maximum resolution can be fully used. Otherwise, the application starts in a window. You can use ALT+ENTER at any time to toggle between the two modes.

Note that if you are in Full Screen mode and the resolution of the window does not match the resolution of the video driver, graphics output will be slow compared to drawing in a window.

**Output**

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
SETWINDOWMENUQQ

Sets a top-level menu as the menu to which a list of current child window names is appended.

Prototype

INTERFACE
  FUNCTION SETWINDOWMENUQQ(MENUID)
    LOGICAL SETWINDOWMENUQQ
    INTEGER(4) MENUID
  END FUNCTION
END INTERFACE

MENUID Input. INTEGER(4). Identifies the menu to hold the child window names, starting with 1 as the leftmost menu.

Description

The list of current child window names can appear in only one menu at a time. If the list of windows is currently in a menu, it is removed from that menu. By default, the list of child windows appears at the end of the Window menu.

Output

The result type is LOGICAL(4). The result is .TRUE. if successful; otherwise, .FALSE..
**SETWSIZEQQ**

*Sets the size and position of a window.*

**Prototype**

```fortran
INTERFACE
    FUNCTION SETWSIZEQQ(IUNIT,WINFO)
        STRUCTURE /QWINFO/
            INTEGER(2) TYPE,X,Y,H,W
        END STRUCTURE
        INTEGER(4) SETWSIZEQQ, IUNIT
        RECORD /QWINFO/ WINFO
        END FUNCTION
    END INTERFACE
```

**IUNIT**  
Input. INTEGER(4). Specifies the window unit. Unit numbers 0, 5, and 6 refer to the default startup window only if the program does not explicitly open them with the OPEN statement. To set the size of the frame window (as opposed to a child window), set IUNIT to the symbolic constant QWIN$FRAMEWINDOW.

**WINFO**  
Input. RECORD /QWINFO/. Physical coordinates of the window's upper-left corner, and the current or maximum height and width of the window's client area (the area within the frame).

```fortran
STRUCTURE /QWINFO/
    INTEGER(2) TYPE ! request type
    INTEGER(2) X ! x coordinate for upper left
    INTEGER(2) Y ! y coordinate for upper left
    INTEGER(2) H ! window height
    INTEGER(2) W ! window width
END STRUCTURE
```
This function's behavior depends on the value of
STRUCTURE /QWININFO/, which can be any of the
following:

- QWIN$MIN: Minimizes the window.
- QWIN$MAX: Maximizes the window.
- QWIN$RESTORE: Restores the minimized window to
  its previous size.
- QWIN$SET: Sets the window's position and
  size according to the other values in qwininfo.

**Description**

The position and dimensions of child windows are expressed in units of
character height and width. The position and dimensions of the frame
window are expressed in screen pixels.

The height and width specified for a frame window reflects the actual size
in pixels of the frame window including any borders, menus, and status bar
at the bottom.

**Output**

The result type is INTEGER(4). The result is zero if successful; otherwise,
nonzero.

---

**UNREGISTERMOUSEEVENT**

*Removes the callback routine registered
for a specified window by an earlier call
to REGISTERMOUSEEVENT.*

**Prototype**

IA-32
INTERFACE
FUNCTION UNREGISTERMOUSEEVENT (UNIT, MouseEvents)
   INTEGER(4) UNREGISTERMOUSEEVENT, UNIT
   ! MS$ ATTRIBUTES C, ALIAS:'unregistermouseeventqq' & :: registermouseevent
   INTEGER(4) MouseEvents
END FUNCTION
END INTERFACE

INTERFACE
FUNCTION REGISTERMOUSEEVENT (UNIT, MouseEvents, & CallBackRoutine)
   INTEGER(4) REGISTERMOUSEEVENT
   ! MS$ ATTRIBUTES C, ALIAS:'unregistermouseeventqq' & :: unregistermouseevent
   INTEGER(8) UNIT
   INTEGER(4) MouseEvents
END FUNCTION
END INTERFACE

UNIT 
Input. INTEGER (4) (IA-32) or INTEGER (8) (Itanium-based systems). Unit number of the window whose callback routine on mouse events is to be unregistered.

MOUSEEVENTS 
Input. INTEGER (4). One or more mouse events handled by the callback routine to be unregistered. Symbolic constants for the possible mouse events are:

- MOUSE$LBUTTONDOWN 
  Left mouse button down
- MOUSE$LBUTTONUP 
  Left mouse button up
- MOUSE$LBUTTONDBLCLK 
  Left mouse button double-click
- MOUSE$RBUTTONDOWN 
  Right mouse button down
MOUSE$RBUTTONUP  Right mouse button up
MOUSE$RBUTTONDBLCLK Right mouse button double-click
MOUSE$MOVE  Mouse moved

Description
Once you call UNREGISTERMOUSEEVENT, QuickWin no longer calls the callback routine specified earlier for the window when mouse events occur. Calling UNREGISTERMOUSEEVENT when no callback routine is registered for the window has no effect.

Output
The result type is INTEGER(4). The result is zero or a positive integer if successful; otherwise, a negative integer which can be one of the following:

MOUSE$BADUNIT  The unit specified is not open, or is not associated with a QuickWin window.
MOUSE$BADEVENT  The event specified is not supported.

WAITONMOUSEEVENT

Waits for the specified mouse input from the user.

Prototype

INTERFACE
FUNCTION WAITONMOUSEEVENT(MouseEvents,KeyState,X,Y)
INTEGER WAITONMOUSEEVENT,MouseEvents,KeyState,X,Y
END FUNCTION
END INTERFACE
MOUSEEVENTS  
Input. INTEGER(4). One or more mouse events that must occur before the function returns. Symbolic constants for the possible mouse events are:

MOUSE$LBUTTONDOWN  
Left mouse button down

MOUSE$LBUTTONUP  
Left mouse button up

MOUSE$LBUTTONDBLCLK  
Left mouse button double-click

MOUSE$RBUTTONDOWN  
Right mouse button down

MOUSE$RBUTTONUP  
Right mouse button up

MOUSE$RBUTTONDBLCLK  
Right mouse button double-click

MOUSE$MOVE  
Mouse moved

KEYSTATE  
Output. INTEGER(4). Bitwise inclusive OR of the state of the mouse during the event. The value returned in KEYSTATE can be any or all of the following symbolic constants:

MOUSE$KS_LBUTTON  
Left mouse button down during event

MOUSE$KS_RBUTTON  
Right mouse button down during event

MOUSE$KS_SHIFT  
SHIFT key held down during event

MOUSE$KS_CONTROL  
CONTROL key held down during event

X  
Output. INTEGER(4). X position of the mouse when the event occurred.

Y  
Output. INTEGER(4). Y position of the mouse when the event occurred.
**Description**

`WAITONMOUSEEVENT` does not return until the specified mouse input is received from the user. While waiting for a mouse event to occur, the status bar changes to read "Mouse input pending in XXX" where XXX is the name of the window. When a mouse event occurs, the status bar returns to its previous value.

A mouse event must happen in the window that had focus when `WAITONMOUSEEVENT` was initially called. Mouse events in other windows will not end the wait. Mouse events in other windows cause callbacks to be called for the other windows, if callbacks were previously registered for those windows.

For every `BUTTONDOWN` or `BUTTONDBLCLK` event there is an associated `BUTTONUP` event. When the user double clicks, four events happen: `BUTTONDOWN` and `BUTTONUP` for the first click, and `BUTTONDBLCLK` and `BUTTONUP` for the second click. The difference between getting `BUTTONDBLCLK` and `BUTTONDOWN` for the second click depends on whether the second click occurs in the double click interval, set in the system's `CONTROL PANEL/MOUSE`.

**Output**

The result type is `INTEGER(4)`. The result is the symbolic constant associated with the mouse event that occurred if successful. If the function fails, it returns the constant `MOUSE$BADEVENT`, meaning the event specified is not supported.

**QuickWin Default Menu Support**

These are predefined callback functions, which you can register as callback routines or call directly from your program.
WINPRINT

Prints the program

Prototype

INTERFACE
SUBROUTINE WINPRINT()
!MS$ ATTRIBUTES stdcall, alias:'_WINPRINT@0' ::
WINPRINT
END SUBROUTINE
END INTERFACE

Description

This callback routine calls “Print...” dialog to print or save as bitmap file contents of current active (focused) child window.

WINSAVE

Saves the program

Prototype

INTERFACE
SUBROUTINE WINSAVE()
!MS$ ATTRIBUTES stdcall, alias:'_WINSAVE@0' ::
WINSAVE
END SUBROUTINE
END INTERFACE

Description

This callback routine calls “Save...” dialog to print or save as bitmap file contents of current active (focused) child window.
WINEXIT

Terminates the program

Prototype

INTERFACE
SUBROUTINE WINEXIT()
!MS$ ATTRIBUTES stdcall, alias:'_WINEXIT@0' ::
WINEXIT
END SUBROUTINE
END INTERFACE

Description

This callback routine calls “Exit...” dialog to terminate the current active (focused) child window.

WINCOPY

Copies the selected text and/or graphics from current window to the Clipboard.

Prototype

INTERFACE
SUBROUTINE WINCOPY()
!MS$ ATTRIBUTES stdcall, alias:'_WINCOPY@0' ::
WINCOPY
END SUBROUTINE
END INTERFACE
Description
This callback routine copies the selected text and/or graphics from current window to the Clipboard.

WINPASTE
Pastes clipboard contents (text only) to the current text window of the active window during a READ.

Prototype
INTERFACE SUBROUTINE WINPASTE()
!MS$ ATTRIBUTES stdcall, alias:'_WINPASTE@0' :: WINPASTE
   END SUBROUTINE
END INTERFACE

Description
This callback routine pastes clipboard contents (text only) to the current text window of the active window during a READ.

WINSIZETOFIT
Sizes output to fit window.

Prototype
INTERFACE SUBROUTINE WINSIZETOFIT()
WINFULLSCREEN

Displays output in full screen.

Prototype

```fortran
INTERFACE
  SUBROUTINE WINFULLSCREEN()
    !MS$ ATTRIBUTES stdcall, alias:'_WINFULLSCREEN@0' ::
    WINFULLSCREEN
  END SUBROUTINE
END INTERFACE
```

Description

This callback routine displays output in full screen.

WINSTATE

Toggles between pause and resume states of text output.

Prototype

```fortran
INTERFACE
```

Description

This callback routine sizes output to fit window.
SUBROUTINE WINSTATE()
!MS$ ATTRIBUTES stdcall, alias:'_WINSTATE@0' ::
WINSTATE
   END SUBROUTINE
   END INTERFACE

Description
This callback routine toggles between pause and resume states of text output.

WINCASCADE
_Cascades active windows._

Prototype
INTERFACE
   SUBROUTINE WINCASCADE()
!MS$ ATTRIBUTES stdcall, alias:'_WINCASCADE@0' ::
WINCASCADE
   END SUBROUTINE
   END INTERFACE

Description
This callback routine cascades active windows.
WIN TILE

Tiles active windows.

Prototype

INTERFACE
  SUBROUTINE WIN TILE()
  !MS$ ATTRIBUTES stdcall, alias:'_WIN TILE@0' ::
  WIN TILE
  END SUBROUTINE
END INTERFACE

Description

This callback routine tiles active windows.

WIN ARRANGE

Arranges icons.

Prototype

INTERFACE
  SUBROUTINE WIN ARRANGE()
  !MS$ ATTRIBUTES stdcall, alias:'_WIN ARRANGE@0' ::
  WIN ARRANGE
  END SUBROUTINE
END INTERFACE

Description

This callback routine arranges icons.
WININPUT

Prototype

INTERFACE
SUBROUTINE WININPUT()
!MS$ ATTRIBUTES stdcall, alias:'_WININPUT@0' ::
WININPUT
   END SUBROUTINE
END INTERFACE

WINCLEARPASTE

Clears the paste buffer.

Prototype

INTERFACE
SUBROUTINE WINCLEARPASTE()
!MS$ ATTRIBUTES stdcall, alias:'_WINCLEARPASTE@0' ::
WINCLEARPASTE
   END SUBROUTINE
END INTERFACE

Description
This callback routine clears the paste buffer.
WINSTATUS

Enables a status bar.

Prototype

INTERFACE
  SUBROUTINE WINSTATUS()
  !MS$ ATTRIBUTES stdcall, alias:'_WINSTATUS@0' ::
  WINSTATUS
  END SUBROUTINE
END INTERFACE

Description
This callback routine enables a status bar.

WININDEX

Displays the index for QuickWin Help.

Prototype

INTERFACE
  SUBROUTINE WININDEX()
  !MS$ ATTRIBUTES stdcall, alias:'_WININDEX@0' ::
  WININDEX
  END SUBROUTINE
END INTERFACE

Description
This callback routine displays the index for QuickWin Help.
WINUSING

Displays information on how to use Help.

Prototype

INTERFACE
SUBROUTINE WINUSING()
!MS$ ATTRIBUTES stdcall, alias:'_WINUSING@0' ::
WINUSING
    END SUBROUTINE
END INTERFACE

Description

This callback routine displays information on how to use help.

WINABOUT

Displays information about the current QuickWin application.

Prototype

INTERFACE
SUBROUTINE WINABOUT()
!MS$ ATTRIBUTES stdcall, alias:'_WINABOUT@0' ::
WINABOUT
    END SUBROUTINE
END INTERFACE
Description
This callback routine displays information about the current QuickWin application.

WINSELECTTEXT
Selects text from the current window.

Prototype
INTERFACE
SUBROUTINE WINSELECTTEXT()
!MS$ ATTRIBUTES stdcall, alias:'_WINSELECTTEXT@0' ::
WINSELECTTEXT
END SUBROUTINE
END INTERFACE

Description
This callback routine selects text from the current window.

WINSELECTGRAPHICS
Selects graphics from the current window.

Prototype
INTERFACE
SUBROUTINE WINSELECTGRAPHICS()
!MS$ ATTRIBUTES stdcall, alias:'_WINSELECTGRAPHICS@0' ::
WINSELECTGRAPHICS
END SUBROUTINE
END INTERFACE
Description
This callback routine selects graphics from the current window.

WINSELECTALL
Selects the entire contents of the current window.

Prototype
INTERFACE
  SUBROUTINE WINSELECTALL()
  !MS$ ATTRIBUTES stdcall, alias:'WINSELECTALL@0' ::
  WINSELECTALL
    END SUBROUTINE
  END INTERFACE

Description
This callback routine selects the entire contents of the current window.

NUL
No callback routine.

Prototype
INTERFACE
  SUBROUTINE NUL()
  !MS$ ATTRIBUTES stdcall, alias:'NUL@0' ::
    END SUBROUTINE
  END INTERFACE
Description
This is a no callback routine. It denies the use of a callback routine.

Unknown Functions

GETACTIVEPAGE

Prototype
INTERFACE
FUNCTION GETACTIVEPAGE()
   INTEGER(2) GETACTIVEPAGE
END FUNCTION
END INTERFACE

GETTEXTCURSOR

Prototype
INTERFACE
FUNCTION GETTEXTCURSOR()
   INTEGER(2) GETTEXTCURSOR
END FUNCTION
END INTERFACE
GETGTEXTVECTOR

Prototype

INTERFACE
  SUBROUTINE GETGTEXTVECTOR(X,Y)
  !MS$ ATTRIBUTES stdcall, ALIAS:"__f_getgtextvector@8"
  :: getgtextvector
    INTEGER(2) X,Y
  !MS$ ATTRIBUTES REFERENCE :: X
  !MS$ ATTRIBUTES REFERENCE :: Y
  END SUBROUTINE
END INTERFACE

GETHANDLEQQ

Prototype

INTERFACE
  FUNCTION GETHANDLEQQ(IUNIT)
  integer*4 GETHANDLEQQ, IUNIT
  END FUNCTION
END INTERFACE
GETVIDEOCONFIG

Prototype

INTERFACE
SUBROUTINE GETVIDEOCONFIG(s)
STRUCTURE /VIDEOCONFIG/
  INTEGER(2) NUMXPIXELS ! number of pixels on X axis
  INTEGER(2) NUMYPIXELS ! number of pixels on Y axis
  INTEGER(2) NUMTEXTCOLS ! number of text columns
    ! available
  INTEGER(2) NUMTEXTROWS ! number of text rows
    ! available
  INTEGER(2) NUMCOLORS ! number of actual colors
  INTEGER(2) BITSPERPIXEL ! number of bits per pixel
  INTEGER(2) NUMVIDEOPAGES ! number of available video pages
  INTEGER(2) MODE ! current video mode
  INTEGER(2) ADAPTER ! active display adapter
  INTEGER(2) MONITOR ! active display monitor
  INTEGER(2) MEMORY ! adapter video memory in K bytes
END STRUCTURE
RECORD /VIDEOCONFIG/ S
!MS$ ATTRIBUTES REFERENCE :: S
END SUBROUTINE
END INTERFACE
GETVISUALPAGE

Prototype

```
INTERFACE
    FUNCTION GETVISUALPAGE()
        INTEGER(2) GETVISUALPAGE
    END FUNCTION
END INTERFACE
```

REGISTERFONTS

Prototype

```
INTERFACE
    FUNCTION REGISTERFONTS(FILENAME)
        INTEGER(2) REGISTERFONTS
    END FUNCTION
END INTERFACE
```

!MS$ ATTRIBUTES ALIAS:"__ILf_registerfonts" ::
REGISTERFONTS

!MS$ ATTRIBUTES REFERENCE :: FILENAME

END FUNCTION
END INTERFACE
SELECTPALETTE

Prototype

INTERFACE
FUNCTION SELECTPALETTE(NUMBER)
  INTEGER(2) SELECTPALETTE, NUMBER
END FUNCTION
END INTERFACE

SETACTIVEPAGE

Prototype

INTERFACE
FUNCTION SETACTIVEPAGE(PAGE)
  INTEGER(2) SETACTIVEPAGE, PAGE
END FUNCTION
END INTERFACE
SETFRAMEWINDOW

Prototype

INTERFACE
  SUBROUTINE SETFRAMEWINDOW(X,Y,WIDTH,HEIGHT)
    INTEGER X,Y,WIDTH,HEIGHT
  END SUBROUTINE
END INTERFACE

SETGTEXTVECTOR

Prototype

INTERFACE
  SUBROUTINE SETGTEXTVECTOR(X,Y)
    INTEGER(2) X,Y
  END SUBROUTINE
END INTERFACE
SETSTATUSMESSAGE

Prototype

INTERFACE
SUBROUTINE SETSTATUSMESSAGE(MSG, ID)
  CHARACTER(LEN=*) MSG
  !MS$ ATTRIBUTES reference :: MSG
  INTEGER(4) ID
END SUBROUTINE
END INTERFACE

SETTEXTCURSOR

Prototype

INTERFACE
FUNCTION SETTEXTCURSOR(ATTR)
  INTEGER(2) SETTEXTCURSOR, ATTR
END FUNCTION
END INTERFACE
SETTEXTFONT

Prototype

INTERFACE
    SUBROUTINE SETTEXTFONT (FONTNAME)
    !MS$ ATTRIBUTES ALIAS:"__ILf_settextfont" ::
    SETTEXTFONT
        character(*) FONTNAME
    !MS$ ATTRIBUTES REFERENCE :: FONTNAME
    END SUBROUTINE
END INTERFACE

SETTEXTROWS

Prototype

INTERFACE
    FUNCTION SETTEXTROWS(ROWS)
        INTEGER(2) SETTEXTROWS,ROWS
    END FUNCTION
END INTERFACE
SETVIDEOMODE

Prototype

INTERFACE
  FUNCTION SETVIDEOMODE(MODE)
    INTEGER(2) SETVIDEOMODE,MODE
  END FUNCTION
END INTERFACE

SETVIDEOMODEROWS

Prototype

INTERFACE
  FUNCTION SETVIDEOMODEROWS(MODE,ROWS)
    INTEGER(2) SETVIDEOMODEROWS
    INTEGER(2) MODE,ROWS
  END FUNCTION
END INTERFACE
SETVISUALPAGE

Prototype

INTERFACE
    FUNCTION SETVISUALPAGE(PAGE)
        INTEGER(2) SETVISUALPAGE, PAGE
    END FUNCTION
END INTERFACE

UNREGISTERFONTS

Prototype

INTERFACE
    SUBROUTINE UNREGISTERFONTS()
    END SUBROUTINE
END INTERFACE
Access to Windows Handles for QuickWin Components

GETHANDLEFRAMEQQ

Prototype

INTERFACE
  FUNCTION GETHANDLEFRAMEQQ()
    INTEGER(4) GETHANDLEFRAMEQQ
  END FUNCTION
END INTERFACE

GETHANDLECLIENTQQ

Prototype

INTERFACE
  FUNCTION GETHANDLECLIENTQQ()
    INTEGER(4) GETHANDLECLIENTQQ
  END FUNCTION
END INTERFACE
GETHANDLECHILDQQ

Prototype

```
INTERFACE
  FUNCTION GETHANDLECHILDQQ(QUICKHND)
    INTEGER(4) GETHANDLECHILDQQ
    INTEGER(4) QUICKHND
  END FUNCTION
END INTERFACE
```

UNUSEDQQ

*Used to avoid “unused” warnings*

Prototype

```
INTERFACE
  SUBROUTINE UNUSEDQQ()
  !MS$ ATTRIBUTES
  C,REFERENCE,VARYING,ALIAS:"__FFunusedqq":UNUSEDQQ
  END SUBROUTINE
END INTERFACE
```

Description

Unused routine: simply returns; used to avoid "unused" warnings.
Index

Symbols
S? environment variable, 2-44
$status environment variable, 2-44

Numerics
4Yposixlib, 3-1

A
ABORT portability function, 2-2
ABOUTBOXQQ QuickWin function, 4-125
ABS intrinsic function, 1-139
ACCESS portability function, 2-3
ACHAR intrinsic function, 1-140
ACOS intrinsic function, 1-141
ACOSD intrinsic function, 1-142
ACOSH intrinsic function, 1-143
actual argument
  intrinsic procedure as, 1-8
ADJUSTL intrinsic function, 1-144
ADJUSTR intrinsic function, 1-145
AIMAG intrinsic function, 1-146
AINT intrinsic function, 1-147
ALARM portability function, 2-5
ALL intrinsic function, 1-148
ALLOCATED intrinsic function, 1-150
AMOD portability function, 2-6
AND intrinsic function, 1-151
ANINT intrinsic function, 1-152
ANY intrinsic function, 1-153
APPENDMENUQQ QuickWin function, 4-126
ARC graphic function, 4-14
ARC_W graphic function, 4-15
argument
  intrinsic procedure as, 1-8
  optional, 1-139
ASIN intrinsic function, 1-155
ASIND intrinsic function, 1-156
ASINH intrinsic function, 1-157
ASSOCIATED intrinsic function, 1-158
ATAN intrinsic function, 1-160
ATAN2 intrinsic function, 1-161
ATAN2D intrinsic function, 1-163
ATAND intrinsic function, 1-164
ATANH intrinsic function, 1-165
attributes
  INTRINSIC, 1-7
availability of intrinsic procedures, 1-2

Index-1
B
BADDRESS intrinsic function, 1-166
BESJ0 portability function, 2-8
BIC, BIS portability functions, 2-10
bit data type
   representation of, 1-11
BIT portability function, 2-10
BIT_SIZE intrinsic function, 1-167
blanks
   padding, 1-231, 1-232, 1-233, 1-234
BSEARCHQQ portability function, 2-11
BTEST intrinsic function, 1-168

C
CDFLOAT portability function, 2-13
CDFLOAT portability subroutine, 2-7
CEILING intrinsic function, 1-169
CHANGEDIRQQ portability function, 2-14
CHANGEDRIVEQQ portability function, 2-14, 2-15
CHAR intrinsic function, 1-170
CHDIR portability function, 2-16
CHMOD portability function, 2-17
CLEARSCREEN graphic subroutine, 4-16
CLEARSTATUSFPQQ portability function, 2-19
CLICKMENUQQ QuickWin function, 4-129
CLOCK portability function, 2-20
CLOCKX portability subroutine, 2-20
CMPLX intrinsic function, 1-171
color functions
   GETBKCOLORRGB, 4-98
   GETCOLORRGB, 4-96
   GETPIXELRGB, 4-99
   GETPIXELRGB_W, 4-101
   GETPIXELSRGB, 4-104
   INTEGERTORGB, 4-112
   RBGTOINTEGER, 4-111
   SETBKCOLORRGB, 4-107
   SETCOLORRGB, 4-105
   SETPIXELRGB, 4-108
   SETPIXELRGB_W, 4-110
   SETPIXELSRGB, 4-102
comment
   INTRINSIC attribute and statement as, 1-7
COMMITQQ portability function, 2-21
COMPL portability function, 2-23
CONJG intrinsic function, 1-172
COS intrinsic function, 1-173
COSD intrinsic function, 1-174
COSH intrinsic function, 1-175
COUNT intrinsic function, 1-176
CSHIPF intrinsic function, 1-179
CSM G portability function, 2-24
CTIME portability function, 2-25

D
data representation model, 1-10
data types
   bit representation, 1-11
   integer representation, 1-11
   real representation, 1-12
   representation of, 1-10
DATE portability subroutine, 2-26
DATE_AND_TIME intrinsic subroutine, 1-181
DATE4 portability subroutine, 2-27
DBESJ0 portability function, 2-28
DBLE intrinsic function, 1-183
DCLOCK portability function, 2-30
DELDIRQ portability function, 2-31
DELETEMENUQQ QuickWin function, 4-130
DELFLEQQ portability function, 2-32
DFLOAT intrinsic function, 1-184
DFLOATI portability function, 2-33
Index

DFLOATJ portability function, 2-34
DFLOATK portability function, 2-35
DIGITS intrinsic function, 1-185
DIM intrinsic function, 1-186
DISPLAYCURSOR graphic function, 4-17
DMOD portability function, 2-35
DNUM intrinsic function, 1-187
DOT_PRODUCT intrinsic function, 1-188
DPROD intrinsic function, 1-189
DRAND portability function, 2-36
DRANDM portability function, 2-37
DRANSET portability subroutine, 2-39
DREAL intrinsic function, 1-190
DSETGTEXTVECTOR QuickWin subroutine, 4-185
DSHIFTL portability function, 2-39
DSHIFTR portability function, 2-40
DTIME portability function, 2-42

E

elemental
  intrinsic function, 1-5
  intrinsic subroutine, 1-3
ELLIPSE_W graphic function, 4-19
ELLIPSE graphic function, 4-18
environment variables
  $?, 2-44
  $status, 2-44
EOSHIFT intrinsic function, 1-192
EPSILON intrinsic function, 1-194
ETIME portability function, 2-43
execution time, computing, 1-248
EXIT portability function, 2-44
EXP intrinsic function, 1-195
EXPONENT intrinsic function, 1-196
extensions
  intrinsic procedures, 1-9
EXTERNAL statement and attribute, 1-2

F

FDATE portability subroutine, 2-45
FFLUSH POSIX subroutine, 3-23
FGETC portability function, 2-46
FGETC POSIX subroutine, 3-24
FINDFILEQQ portability function, 2-47
flib.fd include file, 4-1
FLOATI portability function, 2-111
FLOODFILL graphic function, 4-21
FLOODFILL_W graphic function, 4-22
FLOODFILLRGB graphic function, 4-23
FLOODFILLRGB_W graphic function, 4-24
FLOOR intrinsic function, 1-197
FLUSH portability subroutine, 2-48
FOCUSQQ QuickWin function, 4-131
font manipulation functions
  GETFONTINFO, 4-113
  GETGTEXTEXTENT, 4-115
  GETGTEXTROTATION, 4-121
  GETTEXTCOLORRGB, 4-122
  INITIALIZEFONTS, 4-116
  OUTGTEXT, 4-116
  SETFONT, 4-117
  SETGTEXTROTATION, 4-120
  SETTEXTCOLORRGB, 4-123
FOR_CHECK_FLAWED_PENTIUM
  portability subroutine, 2-49
FOR_GET_FPE portability function, 2-50
FOR_SET_FPE portability function, 2-53
FOR_SET_REENTRANCY portability function, 2-54
FPUTC portability function, 2-51
FPUTC POSIX subroutine, 3-27
FRACTION intrinsic function, 1-198
FREE, 1-199
FSEEK portability subroutine, 2-52
FSEEK POSIX subroutine, 3-28
FSTAT portability subroutine, 2-56
FTELL portability function, 2-58
FTELL POSIX subroutine, 3-30
FULLPATHQQ portability function, 2-59
function
elemental intrinsic, 1-5
generic and specific, 1-4, 1-15
inquiry intrinsic, 1-5
INTERFACE block, 2-1
intrinsic, 1-3
transformational intrinsic, 1-6

GETFILEINFOQQ portability subroutine, 2-79
GETFILLMASK graphic subroutine, 4-31
GETFONTINFO font manipulation function, 4-113
GETGID portability function, 2-83
GETGTESTROTATION font manipulation function, 4-121
GETGTEXTEXTENT font manipulation function, 4-115
GETGTEXTVECTOR QuickWin subroutine, 4-181
GETHANDLECHILDQQ QuickWin window handle function, 4-191
GETHANDLECLIENTQQ QuickWin window handle function, 4-190
GETHANDLEFRAMEQQ QuickWin window handle function, 4-190
GETHANDLEQQ QuickWin function, 4-181
GETHWDQQ QuickWin function, 4-135
GETIM portability subroutine, 2-93
GETIMAGE graphic subroutine, 4-32
GETIMAGE_W graphic subroutine, 4-33
GETIMEOFSAY portability subroutine, 2-94
GETLASTERROR portability function, 2-84
GETLASTERRORQQ portability function, 2-85
GETLINESTYLE graphic function, 4-34
GETLOG portability subroutine, 2-87
GETPHYSCOORD graphic subroutine, 4-35
GETPID portability function, 2-88
GETPIXEL graphic function, 4-36
GETPIXEL_W graphic function, 4-37
GETPIXELRGB color function, 4-99
GETPIXELRGB pixel function, 4-99
GETPIXELRGB_W color function, 4-101
GETPIXELRGB_W pixel function, 4-101
GETPIXELS graphic subroutine, 4-38
GETPIXELSRGB color subroutine, 4-104
GETPIXELSRGB pixel subroutine, 4-104
GETPOS portability function, 2-89
GETSTRQQ portability function, 2-92
GETTEXTCOLOR graphic function, 4-39
GETTEXTCOLORRGB font manipulation function, 4-122
GETTEXTCURSOR QuickWin function, 4-180
GETTEXTPOSITION graphic subroutine, 4-40
GETTEXTWINDOW graphic subroutine, 4-40
GETUID portability function, 2-95
GETUNITQQ QuickWin function, 4-136
GETVIDEOCONFIG QuickWin subroutine, 4-182
GETVIEWCOORD graphic subroutine, 4-41
GETVIEWCOORD_W graphic subroutine, 4-42
GETVISUALPAGE QuickWin function, 4-183
GETWINDOWCONFIG QuickWin function, 4-137
GETWINDOWCOORD graphic subroutine, 4-43
GETWRITEMODE graphic function, 4-44
GETWSIZEQQ QuickWin function, 4-139
GMTIME portability subroutine, 2-96

graphics functions
   ARC, 4-14
   ARC_W, 4-15
   CLEARSCREEN, 4-16
   DISPLAYCURSOR, 4-17
   ELLIPS_W, 4-19
   ELLIPSE, 4-18
   FLOODFILL, 4-21
   FLOODFILL_W, 4-22
   FLOODFILLRGB, 4-23
   FLOODFILLRGB_W, 4-24
   GETARCINFO, 4-25
   GETBKCOLOR, 4-27
   GETCOLOR, 4-28
   GETCURRENTPOSITION, 4-29
   GETCURRENTPOSITION_W, 4-30
   GETFILLMASK, 4-31
   GETIMAGE, 4-32
   GETIMAGE_W, 4-33
GETLINESTYLE, 4-34
GETPHYSCOORD, 4-35
GETPIXEL, 4-36
GETPIXEL_W, 4-37
GETPIXELS, 4-38
GETTEXTCOLOR, 4-39
GETTEXTEXPOSITION, 4-40
GETTEXTEXTWINDOW, 4-40
GETVIEWCOORD, 4-41
GETVIEWCOORD_W, 4-42
GETWINDOWCOORD, 4-43
GETWRITEMODE, 4-44
GRSTATUS, 4-45
IMAGESIZE, 4-46
IMAGESIZE_W, 4-47
LINETO, 4-48
LINETO_W, 4-49
LOADIMAGE, 4-52
LOADIMAGE_W, 4-53
MOVETO, 4-54
MOVETO_W, 4-55
OUTTEXT, 4-56
PIE, 4-56
PIE_W, 4-58
POLYGON, 4-60
POLYGON_W, 4-61
PUTIMAGE, 4-63
PUTIMAGE_W, 4-65
RECTANGLE, 4-68
RECTANGLE_W, 4-69
REMAPALLPALETTERGB, 4-70
REMAPALLPALETTERGP, 4-70
REMAPPALETTERGB, 4-72
SAVEIMAGE, 4-74
SAVEIMAGE_W, 4-75
SCROLLTEXTWINDOW, 4-78
SETBKCOLOR, 4-79
SETCLIPRGN, 4-80
SETCOLOR, 4-81
SETFILLMASK, 4-82
SETLINESTYLE, 4-83
SETPIXEL, 4-84
SETPIXEL_W, 4-85
SETPIXELS, 4-86
SETTEXTCOLOR, 4-87
SETTEXTPOSITION, 4-88
SETTEXTWINDOW, 4-89
SETVIEWORG, 4-90
SETVIEWPORT, 4-91
SETWINDOW, 4-92
SETWRITEMODE, 4-93
WRAPON, 4-95

graphics procedures, 4-5, 4-6
GRSTATUS graphic function, 4-45

H
HFIX intrinsic function, 1-200
HOSTNAM portability function, 2-97
HOSTNM portability subroutine, 2-98
HUGE intrinsic function, 1-201

I
IACHAR intrinsic function, 1-202
IADDR intrinsic function, 1-203
IAND intrinsic function, 1-204
IARG portability function, 2-99
IARGC portability function, 2-99, 2-100
IBCLR intrinsic function, 1-205
IBITS intrinsic function, 1-206
IBSET intrinsic function, 1-207
ICHAR intrinsic function, 1-208
IDATE portability subroutine, 2-101
IDATE4 portability subroutine, 2-102
IDFLOAT portability function, 2-103
IDIM intrinsic function, 1-209
IEEE_FLAGS portability function, 2-104
IEEE_HANDLER portability function, 2-108
IEOR intrinsic function, 1-210
IERNNO portability function, 2-109
IFL_RUNTIME_INIT portability subroutine, 2-110
IFLOATI portability function, 2-112
IFLOATJ portability functions, 2-112
IFLPORT.F90, 2-2, 2-199
IJINT intrinsic function, 1-211
IMAG intrinsic function, 1-212
IMAGESIZE graphic function, 4-46
IMAGESIZE_W graphic function, 4-47
IMOD portability functions, 2-113
INDEX intrinsic function, 1-213
INITIALIZEFONTS font manipulation function, 4-116
INMAX portability function, 2-114
INQFOCUSQQ QuickWin function, 4-141
inquiry function, 1-5
INSERTMENUQQ QuickWin function, 4-143
INT intrinsic function, 1-214
INT1 intrinsic function, 1-215
INT2 intrinsic function, 1-216
INT4 intrinsic function, 1-217
INT8 intrinsic function, 1-217
INTC portability function, 2-114
integer
representation of, 1-11
INTEGRERTORGB color subroutine, 4-112
INTERFACE block, 2-1
INTERFACE portability function, 2-1
intrinsic
functions, 1-1, 1-3
procedures, 1-1
subroutines, 1-3
INTRINSIC attribute and statement, 1-7
intrinsic procedures
ABS, 1-139
ACHAR, 1-140
ACOS, 1-141
ACOSD, 1-142
ACOSH, 1-143
ADJSTL, 1-144
ADJSTR, 1-145
<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1-146</td>
</tr>
<tr>
<td>AIMAG</td>
<td>1-146</td>
</tr>
<tr>
<td>AINT</td>
<td>1-147</td>
</tr>
<tr>
<td>ALL</td>
<td>1-148</td>
</tr>
<tr>
<td>ALLOCATED</td>
<td>1-150</td>
</tr>
<tr>
<td>AND</td>
<td>1-151</td>
</tr>
<tr>
<td>ANINT</td>
<td>1-152</td>
</tr>
<tr>
<td>ANY</td>
<td>1-153</td>
</tr>
<tr>
<td>ASIN</td>
<td>1-155</td>
</tr>
<tr>
<td>ASIND</td>
<td>1-156</td>
</tr>
<tr>
<td>ASINH</td>
<td>1-157</td>
</tr>
<tr>
<td>ASSOCIATED</td>
<td>1-158</td>
</tr>
<tr>
<td>ATAN</td>
<td>1-160</td>
</tr>
<tr>
<td>ATAN2</td>
<td>1-161</td>
</tr>
<tr>
<td>ATAN2D</td>
<td>1-163</td>
</tr>
<tr>
<td>ATAND</td>
<td>1-164</td>
</tr>
<tr>
<td>ATANH</td>
<td>1-165</td>
</tr>
<tr>
<td>availability</td>
<td>1-2</td>
</tr>
<tr>
<td>BADDRESS</td>
<td>1-166</td>
</tr>
<tr>
<td>BIT_SIZE</td>
<td>1-167</td>
</tr>
<tr>
<td>BTEST</td>
<td>1-168</td>
</tr>
<tr>
<td>CEILING</td>
<td>1-169</td>
</tr>
<tr>
<td>CHAR</td>
<td>1-170</td>
</tr>
<tr>
<td>classes of</td>
<td>1-2</td>
</tr>
<tr>
<td>CMPLX</td>
<td>1-171</td>
</tr>
<tr>
<td>CONJG</td>
<td>1-172</td>
</tr>
<tr>
<td>COS</td>
<td>1-173</td>
</tr>
<tr>
<td>COSD</td>
<td>1-174</td>
</tr>
<tr>
<td>COSH</td>
<td>1-175</td>
</tr>
<tr>
<td>COUNT</td>
<td>1-176</td>
</tr>
<tr>
<td>CSHIFT</td>
<td>1-179</td>
</tr>
<tr>
<td>data type representation</td>
<td>1-10</td>
</tr>
<tr>
<td>DATE_AND_TIME</td>
<td>1-181</td>
</tr>
<tr>
<td>DBLE</td>
<td>1-183</td>
</tr>
<tr>
<td>DFLOAT</td>
<td>1-184</td>
</tr>
<tr>
<td>DIGITS</td>
<td>1-185</td>
</tr>
<tr>
<td>DIM</td>
<td>1-186</td>
</tr>
<tr>
<td>DNUM</td>
<td>1-187</td>
</tr>
<tr>
<td>DOT_PRODUCT</td>
<td>1-188</td>
</tr>
<tr>
<td>DPROD</td>
<td>1-189</td>
</tr>
<tr>
<td>DREAL</td>
<td>1-190</td>
</tr>
<tr>
<td>elemental function</td>
<td>1-5</td>
</tr>
<tr>
<td>elemental subroutine</td>
<td>1-3</td>
</tr>
<tr>
<td>EOSHIFT</td>
<td>1-192</td>
</tr>
<tr>
<td>EPSILON</td>
<td>1-194</td>
</tr>
<tr>
<td>EXP</td>
<td>1-195</td>
</tr>
<tr>
<td>EXPONENT</td>
<td>1-196</td>
</tr>
<tr>
<td>FLOOR</td>
<td>1-197</td>
</tr>
<tr>
<td>FRACTION</td>
<td>1-198</td>
</tr>
<tr>
<td>FREE</td>
<td>1-199</td>
</tr>
<tr>
<td>functions</td>
<td>1-3</td>
</tr>
<tr>
<td>generic and specific</td>
<td>1-4, 1-15</td>
</tr>
<tr>
<td>HFIX</td>
<td>1-199, 1-200</td>
</tr>
<tr>
<td>HUGE</td>
<td>1-201</td>
</tr>
<tr>
<td>IADDRESS</td>
<td>1-202</td>
</tr>
<tr>
<td>IAND</td>
<td>1-204</td>
</tr>
<tr>
<td>IBCLR</td>
<td>1-205</td>
</tr>
<tr>
<td>IBITS</td>
<td>1-206</td>
</tr>
<tr>
<td>IBSET</td>
<td>1-207</td>
</tr>
<tr>
<td>ICHAR</td>
<td>1-208</td>
</tr>
<tr>
<td>IDIM</td>
<td>1-209</td>
</tr>
<tr>
<td>IEOR</td>
<td>1-210</td>
</tr>
<tr>
<td>IJINT</td>
<td>1-211</td>
</tr>
<tr>
<td>IMAG</td>
<td>1-212</td>
</tr>
<tr>
<td>INDEX</td>
<td>1-213</td>
</tr>
<tr>
<td>inquiry function</td>
<td>1-5</td>
</tr>
<tr>
<td>INT</td>
<td>1-214</td>
</tr>
<tr>
<td>INT1</td>
<td>1-215</td>
</tr>
<tr>
<td>INT2</td>
<td>1-216</td>
</tr>
<tr>
<td>INT4</td>
<td>1-217</td>
</tr>
<tr>
<td>INT8</td>
<td>1-217</td>
</tr>
<tr>
<td>INUM</td>
<td>1-218</td>
</tr>
<tr>
<td>INTRINSIC attribute</td>
<td>1-7</td>
</tr>
<tr>
<td>INTRINSIC statement</td>
<td>1-7</td>
</tr>
<tr>
<td>IPINT</td>
<td>1-219</td>
</tr>
<tr>
<td>IQINT</td>
<td>1-220</td>
</tr>
<tr>
<td>ISHIFT</td>
<td>1-220</td>
</tr>
<tr>
<td>ISHFTC</td>
<td>1-221</td>
</tr>
<tr>
<td>ISIGN</td>
<td>1-223</td>
</tr>
<tr>
<td>ISNAN</td>
<td>1-224</td>
</tr>
<tr>
<td>IXOR</td>
<td>1-224</td>
</tr>
<tr>
<td>JNUM</td>
<td>1-226</td>
</tr>
<tr>
<td>keywords</td>
<td>1-139</td>
</tr>
<tr>
<td>KIND</td>
<td>1-226</td>
</tr>
<tr>
<td>LBOUND</td>
<td>1-227</td>
</tr>
<tr>
<td>LEN</td>
<td>1-229</td>
</tr>
<tr>
<td>LEN_TRIM</td>
<td>1-230</td>
</tr>
</tbody>
</table>
Index-8

INUM intrinsic function, 1-218
IOR intrinsic function, 1-219
IPXFARGC POSIX function, 3-2
IQINT intrinsic function, 1-220
IRAND portability function, 2-115
IRANDM portability function, 2-116

INUM intrinsic function, 1-218
IOR intrinsic function, 1-219
IPXFARGC POSIX function, 3-2
IQINT intrinsic function, 1-220
IRAND portability function, 2-115
IRANDM portability function, 2-116
IRANGERT portability subroutine, 2-116
IRANSET portability subroutine, 2-117
ISATTY portability function, 2-117
ISHFT intrinsic function, 1-220
ISHFTC intrinsic function, 1-221
ISIGN intrinsic function, 1-223
ISNAN intrinsic function, 1-224
ITIME portability subroutine, 2-118
IXOR intrinsic function, 1-224

J
JABS, 2-119
JABS portability function, 2-119
JDATE portability subroutine, 2-119
JDATE4 portability subroutine, 2-120
JNUM intrinsic function, 1-226

K
keywords
in intrinsic procedures, 1-139
KILL portability function, 2-121
KIND intrinsic function, 1-226

L
LBOUND intrinsic function, 1-227
LCWRQ portability subroutine, 2-122
LEADZ portability function, 2-123
LEN intrinsic function, 1-229
LEN_TRIM intrinsic function, 1-230
LGE intrinsic function, 1-231
LGT intrinsic function, 1-232
libcl.a library, 1-1
libF90.a library, 1-1
libPEPCF90.lib library, 2-1
libPOSF90.lib library, 3-1
libQWF90.lib library, 4-1
LINETO graphic function, 4-48
LINETO_W graphic function, 4-49
LLE intrinsic function, 1-233
LLT intrinsic function, 1-234
LNBLNK portability function, 2-124
LOADIMAGE graphic function, 4-52
LOADIMAGE_W graphic function, 4-53
LOC intrinsic function, 1-235
Locale Formatting Procedures, 2-223
LOG intrinsic function, 1-235
LOG10 intrinsic function, 1-236
LOGICAL intrinsic function, 1-237
LONG portability function, 2-125
LSHFT intrinsic function, 1-238
LSHIFT intrinsic function, 1-238
LSTAT portability function, 2-125
LTIME portability subroutine, 2-126

M
MAKEDIRQQ portability function, 2-128
MALLOC, 1-239
MATHERRQQ portability subroutine, 2-129
MATMUL intrinsic function, 1-240
MAX intrinsic function, 1-242
MAXEXPONENT intrinsic function, 1-243
MAXLOC intrinsic function, 1-244
MAXVAL intrinsic function, 1-246
MBCharLen MBCS inquiry function, 2-201, 2-230
MBConvertMBToUnicode MBCS conversion function, 2-202, 2-237
MBConvertUnicodeToMB MBCS conversion function, 2-202, 2-239
MBCS Conversion Procedures, 2-237
MBCS conversion procedures
   MBConvertMBToUnicode, 2-237
   MBConvertUnicodeToMB, 2-239
MBCS Fortran Equivalent Procedures, 2-241
MBCS Fortran equivalent procedures
MBINCHARQQ, 2-241
MBINDEX, 2-242
MBJISTToJMS, 2-248
MBJISTToJIS, 2-249
MBLEQ, 2-243
MBLGE, 2-243
MBLGT, 2-243
MBLL, 2-243
MBLN, 2-243
MBSCAN, 2-246
MBVERIFY, 2-247
MBCS Inquiry Procedures, 2-230
MBCS inquiry procedures
MBCharLen, 2-230
MBCurMax, 2-231
MBLen, 2-232
MBLen_Trim, 2-233
MBNext, 2-234
MBPrev, 2-235
MBStrLead, 2-236
MBCurMax MBCS inquiry function, 2-201, 2-231
MBINCHARQQ MBCS Fortran equivalent function, 2-203, 2-241
MBINDEX MBCS Fortran equivalent function, 2-203, 2-242
MBJISTToJMS MBCS Fortran equivalent function, 2-203, 2-248
MBJISTToJIS MBCS Fortran equivalent function, 2-203, 2-249
MBLead MBCS inquiry function, 2-201
MBLen MBCS inquiry function, 2-201, 2-232
MBLen_Trim MBCS inquiry function, 2-202, 2-233
MBLEQ MBCS Fortran equivalent function, 2-203, 2-243
MBLGE MBCS Fortran equivalent function, 2-203, 2-243
MBLGT MBCS Fortran equivalent function, 2-203, 2-243
MBLLLE MBCS Fortran equivalent function, 2-203, 2-243
MBLLT MBCS Fortran equivalent function, 2-203, 2-243
MBLINE MBCS Fortran equivalent function, 2-203, 2-243
MBNext MBCS inquiry function, 2-202, 2-234
MBPrev MBCS inquiry function, 2-202, 2-235
MBSCAN MBCS Fortran equivalent function, 2-203, 2-246
MBStrLead MBCS inquiry function, 2-202, 2-236
MBVERIFY MBCS Fortran equivalent function, 2-203, 2-247
MCLOCK intrinsic function, 1-248
measuring program speed, 1-248
MERGE intrinsic function, 1-249
MESSAGEBOXQQ QuickWin function, 4-146
Microsoft Visual C++ 32-bit edition for Windows, xxiv
MIN intrinsic function, 1-250
MINEXPONENT intrinsic function, 1-251
MINLOC intrinsic function, 1-252
MINVAL intrinsic function, 1-254
MKDIR POSIX subroutine, 3-49
MOD intrinsic function, 1-256
MODIFYMENUFLAGSQQ QuickWin function, 4-148
MODIFYMENUROUTINEQQ QuickWin function, 4-149
MODIFYMENUSTRINGQQ QuickWin function, 4-151
MODULO intrinsic function, 1-257
MOVETO graphic subroutine, 4-54
MOVETO_W graphic subroutine, 4-55
MVBITS intrinsic subroutine, 1-258
## N

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>names</td>
<td>1-2</td>
</tr>
<tr>
<td>NaN (not a number)</td>
<td>1-224</td>
</tr>
<tr>
<td>NARGS portability function</td>
<td>2-132</td>
</tr>
<tr>
<td>National Language Support routines</td>
<td>2-199</td>
</tr>
<tr>
<td>NEAREST intrinsic function</td>
<td>1-259</td>
</tr>
<tr>
<td>NINT intrinsic function</td>
<td>1-260</td>
</tr>
<tr>
<td>NLS Locale Formatting Procedures</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSFormatCurrency</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSFormatDate</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSFormatNumber</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSFormatTime</td>
<td>2-201</td>
</tr>
<tr>
<td>NLS Locale Setting and Inquiry Procedures</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSGetEnvironmentCodepage</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSGetLocale</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSGetLocaleInfo</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSSetEnvironmentCodepage</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSSetLocale</td>
<td>2-200</td>
</tr>
<tr>
<td>NLS MBCS Conversion Procedures</td>
<td>2-202</td>
</tr>
<tr>
<td>MBConvertMBToUnicode</td>
<td>2-202</td>
</tr>
<tr>
<td>MBConvertUnicodeToMB</td>
<td>2-202</td>
</tr>
<tr>
<td>NLS MBCS Fortran Equivalent Procedures</td>
<td>2-203</td>
</tr>
<tr>
<td>MBINCHARQQ</td>
<td>2-203</td>
</tr>
<tr>
<td>MBINDEX</td>
<td>2-203</td>
</tr>
<tr>
<td>MBJSTToJMS</td>
<td>2-203</td>
</tr>
<tr>
<td>MBJSTToJIX</td>
<td>2-203</td>
</tr>
<tr>
<td>MBLEQ</td>
<td>2-203</td>
</tr>
<tr>
<td>MBLGE</td>
<td>2-203</td>
</tr>
<tr>
<td>MBLGT</td>
<td>2-203</td>
</tr>
<tr>
<td>MBLLE</td>
<td>2-203</td>
</tr>
<tr>
<td>MBLLT</td>
<td>2-203</td>
</tr>
<tr>
<td>MBLNE</td>
<td>2-203</td>
</tr>
<tr>
<td>MBSCAN</td>
<td>2-203</td>
</tr>
<tr>
<td>MBVERIFY</td>
<td>2-203</td>
</tr>
<tr>
<td>NLS MBCS Inquiry Procedures</td>
<td>2-201</td>
</tr>
<tr>
<td>MBCharLen</td>
<td>2-201</td>
</tr>
<tr>
<td>MBCurMax</td>
<td>2-201</td>
</tr>
<tr>
<td>MBLlead</td>
<td>2-201</td>
</tr>
<tr>
<td>MBLen</td>
<td>2-201</td>
</tr>
<tr>
<td>MBLen_Trim</td>
<td>2-202</td>
</tr>
<tr>
<td>MBNext</td>
<td>2-202</td>
</tr>
<tr>
<td>MBPrev</td>
<td>2-202</td>
</tr>
<tr>
<td>MBStrLead</td>
<td>2-202</td>
</tr>
<tr>
<td>NLS Multi-byte Routines and Functions</td>
<td>2-200</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
<tr>
<td>Locale Formatting</td>
<td></td>
</tr>
<tr>
<td>Locale Setting and Inquiry</td>
<td></td>
</tr>
<tr>
<td>MBCS Conversion</td>
<td></td>
</tr>
<tr>
<td>MBCS Fortran Equivalent Procedures</td>
<td></td>
</tr>
<tr>
<td>MBCS Inquiry</td>
<td></td>
</tr>
<tr>
<td>NLS procedures</td>
<td></td>
</tr>
<tr>
<td>NLSEnumCodepages function</td>
<td>2-204</td>
</tr>
<tr>
<td>NLSEnumLocales function</td>
<td>2-205</td>
</tr>
<tr>
<td>NLSFormatCurrency</td>
<td>2-223</td>
</tr>
<tr>
<td>NLSFormatDate</td>
<td>2-225</td>
</tr>
<tr>
<td>NLSFormatNumber</td>
<td>2-226</td>
</tr>
<tr>
<td>NLSFormatTime</td>
<td>2-228</td>
</tr>
<tr>
<td>NLSGetEnvironmentCodepage</td>
<td>2-206</td>
</tr>
<tr>
<td>NLSGetLocale</td>
<td>2-207</td>
</tr>
<tr>
<td>NLSGetLocaleInfo</td>
<td>2-208</td>
</tr>
<tr>
<td>NLSGetEnvironmentCodepage</td>
<td>2-219</td>
</tr>
<tr>
<td>NLSSetLocale</td>
<td>2-221</td>
</tr>
<tr>
<td>NLS routines</td>
<td>2-199</td>
</tr>
<tr>
<td>NLSEnumCodepages inquiry function</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSEnumLocales inquiry function</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSFormatCurrency locale formatting function</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSFormatDate locale formatting function</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSFormatNumber locale formatting function</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSFormatTime locale formatting function</td>
<td>2-201</td>
</tr>
<tr>
<td>NLSGetEnvironmentCodepage inquiry function</td>
<td>2-200</td>
</tr>
<tr>
<td>NLSGetLocale inquiry subroutine</td>
<td>2-200</td>
</tr>
</tbody>
</table>
NLSGetLocale NLS subroutine, 2-207
NLSGetLocaleInfo inquiry function, 2-200, 2-208
NLSSetEnvironmentCodepage inquiry function, 2-200
NLSSetEnvironmentCodepage inquiry function, 2-219
NLSSetLocale inquiry function, 2-200, 2-221
nonstandard intrinsic procedure, 1-9, 1-15
NOT intrinsic function, 1-261
NUL QuickWin menu subroutine, 4-179
NUMARG portability function, 2-133

O

optional argument, 1-139
OR intrinsic function, 1-262
portability functions
   CHMOD, 2-17
   CLEARSTATUSFPQQ, 2-19
OUTGTEXT font manipulation subroutine, 4-116
OUTTEXT graphic subroutine, 4-56

P

PACK intrinsic function, 1-263
PACKTIMEQQ portability subroutine, 2-134
padding
   blank, 1-231, 1-232, 1-233, 1-234
PEEKCHARQQ portability function, 2-136
PERERROR portability subroutine, 2-137
PIE graphic function, 4-56
PIE_W graphic function, 4-58
pixel functions
   GETPIXELRGB, 4-99
   GETPIXELRGB_W, 4-101
   GETPIXELSRGB, 4-104
   SETPIXELRGB, 4-108
   SETPIXELRGB_W, 4-110
   SETPIXELSRGB, 4-102
   POLYGON graphic function, 4-60
   POLYGON_W graphic function, 4-61
   POPCNT portability function, 2-138
   POPPAR portability function, 2-138
   portability and nonstandard intrinsic procedures, 1-9
   portability functions, 2-119
   /Yportlib, 2-1
   ABORT, 2-2
   ACCESS, 2-3
   ALARM, 2-5
   AMOD, 2-6
   BEEPQQ, 2-7
   BESJ0, 2-8
   BIC, BIS, 2-10
   BIT, 2-10
   BSEARCHQQ, 2-11
   CDFLOAT, 2-13
   CHANGEDIRQQ, 2-14
   CHANGEDRIVEQQ, 2-14, 2-15
   CHDIR, 2-16
   CLOCK, 2-20
   CLOCKX, 2-20
   COMMITQQ, 2-21
   COMPL, 2-23
   CTIME, 2-25
   DATE, 2-26
   DATE4, 2-27
   DBESJ0, 2-28
   DCLOCK, 2-30
   DELDIRQQ, 2-31
   DELFILESQQ, 2-32
   DFFFFFF, 2-32
   Dfloati, 2-33
   Dfloati, 2-34
   Dfloatk, 2-35
   DMOD, 2-35
   DRAND, 2-36
   DRANDM, 2-37
   DRANSET, 2-39
   DSHIFTL, 2-39
   DSHIFTR, 2-40
   DTIME, 2-42
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETIME</td>
<td>2-43</td>
</tr>
<tr>
<td>EXIT</td>
<td>2-44</td>
</tr>
<tr>
<td>FDATE</td>
<td>2-45</td>
</tr>
<tr>
<td>FGETC</td>
<td>2-46</td>
</tr>
<tr>
<td>FINDFILEQQ</td>
<td>2-47</td>
</tr>
<tr>
<td>FLOAII, 2-111</td>
<td></td>
</tr>
<tr>
<td>FLUSH</td>
<td>2-48</td>
</tr>
<tr>
<td>FOR_CHECK_FLADED_PENTIUM</td>
<td>2-49</td>
</tr>
<tr>
<td>FOR_GET_FPE</td>
<td>2-50</td>
</tr>
<tr>
<td>FOR_SET_FPE</td>
<td>2-53</td>
</tr>
<tr>
<td>FOR_SET_REENTRANCY</td>
<td>2-54</td>
</tr>
<tr>
<td>FPUTC</td>
<td>2-51</td>
</tr>
<tr>
<td>FSEEK</td>
<td>2-52</td>
</tr>
<tr>
<td>FSTAT</td>
<td>2-56</td>
</tr>
<tr>
<td>FTEND</td>
<td>2-58</td>
</tr>
<tr>
<td>FULLPATHQQ</td>
<td>2-59</td>
</tr>
<tr>
<td>GERRNO</td>
<td>2-61</td>
</tr>
<tr>
<td>GETARG</td>
<td>2-62</td>
</tr>
<tr>
<td>GETC</td>
<td>2-64</td>
</tr>
<tr>
<td>GETCHARQQ</td>
<td>2-65</td>
</tr>
<tr>
<td>GETCONTROLFPQQ</td>
<td>2-66</td>
</tr>
<tr>
<td>GETCWD</td>
<td>2-69</td>
</tr>
<tr>
<td>GETDAT</td>
<td>2-70</td>
</tr>
<tr>
<td>GETDRIVEDIRQQ</td>
<td>2-71</td>
</tr>
<tr>
<td>GETDRIVESIZEQQ</td>
<td>2-73</td>
</tr>
<tr>
<td>GETDRIVESQQ</td>
<td>2-75</td>
</tr>
<tr>
<td>GETENV</td>
<td>2-76</td>
</tr>
<tr>
<td>GETENVQQ</td>
<td>2-77</td>
</tr>
<tr>
<td>GETFILEINFOQQ</td>
<td>2-79</td>
</tr>
<tr>
<td>GETGID</td>
<td>2-83</td>
</tr>
<tr>
<td>GETLASTERROR</td>
<td>2-84</td>
</tr>
<tr>
<td>GETLASTERRORROQQ</td>
<td>2-85</td>
</tr>
<tr>
<td>GETLOG</td>
<td>2-87</td>
</tr>
<tr>
<td>GETPID</td>
<td>2-88</td>
</tr>
<tr>
<td>GETPOS</td>
<td>2-89</td>
</tr>
<tr>
<td>GETSTROQ</td>
<td>2-92</td>
</tr>
<tr>
<td>GETTIM</td>
<td>2-93</td>
</tr>
<tr>
<td>GETTIMEOFDAY</td>
<td>2-94</td>
</tr>
<tr>
<td>GETUID</td>
<td>2-95</td>
</tr>
<tr>
<td>GMTIME</td>
<td>2-96</td>
</tr>
<tr>
<td>HOSTNM</td>
<td>2-97</td>
</tr>
<tr>
<td>IARG</td>
<td>2-99</td>
</tr>
<tr>
<td>IARGC</td>
<td>2-99, 2-100</td>
</tr>
<tr>
<td>IDATE</td>
<td>2-101</td>
</tr>
<tr>
<td>IDATE4</td>
<td>2-102</td>
</tr>
<tr>
<td>IDFLOAT</td>
<td>2-103</td>
</tr>
<tr>
<td>IEEF_FLAGS</td>
<td>2-104</td>
</tr>
<tr>
<td>IEEF_HANDLER</td>
<td>2-108</td>
</tr>
<tr>
<td>IERRNO</td>
<td>2-109</td>
</tr>
<tr>
<td>IFL_RUNTIME_INIT</td>
<td>2-110</td>
</tr>
<tr>
<td>IFLOATI</td>
<td>2-112</td>
</tr>
<tr>
<td>IFLOATI</td>
<td>2-112</td>
</tr>
<tr>
<td>IFPORT.F90</td>
<td>2-2</td>
</tr>
<tr>
<td>IMOD</td>
<td>2-113</td>
</tr>
<tr>
<td>INMAX</td>
<td>2-114</td>
</tr>
<tr>
<td>INTC</td>
<td>2-114</td>
</tr>
<tr>
<td>INTERFACE block</td>
<td>2-1</td>
</tr>
<tr>
<td>IRAND</td>
<td>2-115</td>
</tr>
<tr>
<td>IRANDM</td>
<td>2-116</td>
</tr>
<tr>
<td>IRANGET</td>
<td>2-116</td>
</tr>
<tr>
<td>IRANSET</td>
<td>2-117</td>
</tr>
<tr>
<td>ISATTY</td>
<td>2-117</td>
</tr>
<tr>
<td>ITIME</td>
<td>2-118</td>
</tr>
<tr>
<td>JDATE</td>
<td>2-119</td>
</tr>
<tr>
<td>JDATE4</td>
<td>2-120</td>
</tr>
<tr>
<td>KILL</td>
<td>2-121</td>
</tr>
<tr>
<td>LCWRQQ</td>
<td>2-122</td>
</tr>
<tr>
<td>LEADZ</td>
<td>2-123</td>
</tr>
<tr>
<td>LNBLNK</td>
<td>2-124</td>
</tr>
<tr>
<td>LONG</td>
<td>2-125</td>
</tr>
<tr>
<td>LSTATG</td>
<td>2-125</td>
</tr>
<tr>
<td>LTIME</td>
<td>2-126</td>
</tr>
<tr>
<td>MAKEDIRQQ</td>
<td>2-128</td>
</tr>
<tr>
<td>MATHERRQQ</td>
<td>2-129</td>
</tr>
<tr>
<td>NARGS</td>
<td>2-132</td>
</tr>
<tr>
<td>NUMARG</td>
<td>2-133</td>
</tr>
<tr>
<td>PACKTIMEQQ</td>
<td>2-134</td>
</tr>
<tr>
<td>PEEKCHARQQ</td>
<td>2-136</td>
</tr>
<tr>
<td>PERROR</td>
<td>2-137</td>
</tr>
<tr>
<td>POPCNT</td>
<td>2-138</td>
</tr>
<tr>
<td>POPPAR</td>
<td>2-138</td>
</tr>
<tr>
<td>PUTC</td>
<td>2-139</td>
</tr>
<tr>
<td>QRAANSET</td>
<td>2-140</td>
</tr>
<tr>
<td>QSORT</td>
<td>2-140</td>
</tr>
<tr>
<td>RAISEQQ</td>
<td>2-141</td>
</tr>
<tr>
<td>RAN</td>
<td>2-143</td>
</tr>
<tr>
<td>RAND</td>
<td>2-144</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>RANDOM</td>
<td>2-146</td>
</tr>
<tr>
<td>RANDU</td>
<td>2-147</td>
</tr>
<tr>
<td>RANF</td>
<td>2-148</td>
</tr>
<tr>
<td>RANGET</td>
<td>2-149</td>
</tr>
<tr>
<td>RANSET</td>
<td>2-149</td>
</tr>
<tr>
<td>RENAME</td>
<td>2-150</td>
</tr>
<tr>
<td>RENAMEFILEQQ</td>
<td>2-151</td>
</tr>
<tr>
<td>RINDEX</td>
<td>2-152</td>
</tr>
<tr>
<td>RTC</td>
<td>2-153</td>
</tr>
<tr>
<td>RUNQQ</td>
<td>2-154</td>
</tr>
<tr>
<td>SCANENV</td>
<td>2-156</td>
</tr>
<tr>
<td>SCWRQQ</td>
<td>2-155</td>
</tr>
<tr>
<td>SECNDS</td>
<td>2-157</td>
</tr>
<tr>
<td>SEED</td>
<td>2-157</td>
</tr>
<tr>
<td>SETCONTROLFPQQ</td>
<td>2-159</td>
</tr>
<tr>
<td>SETDAT</td>
<td>2-161</td>
</tr>
<tr>
<td>SETENVQQ</td>
<td>2-162</td>
</tr>
<tr>
<td>SETERRORMODEQQ</td>
<td>2-164</td>
</tr>
<tr>
<td>SETFILEACCESSQQ</td>
<td>2-166</td>
</tr>
<tr>
<td>SETFILENAMEQQ</td>
<td>2-167</td>
</tr>
<tr>
<td>SETTIM</td>
<td>2-169</td>
</tr>
<tr>
<td>SHIFTL</td>
<td>2-170</td>
</tr>
<tr>
<td>SHIFTR</td>
<td>2-171</td>
</tr>
<tr>
<td>SHORT</td>
<td>2-172</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>2-173</td>
</tr>
<tr>
<td>SIGNALQQ</td>
<td>2-176</td>
</tr>
<tr>
<td>SLEEP</td>
<td>2-179</td>
</tr>
<tr>
<td>SLEEPQQ</td>
<td>2-179</td>
</tr>
<tr>
<td>SORTQQ</td>
<td>2-180</td>
</tr>
<tr>
<td>SPLITPATHQQ</td>
<td>2-182</td>
</tr>
<tr>
<td>SRAND</td>
<td>2-184</td>
</tr>
<tr>
<td>SSWRQQ</td>
<td>2-185</td>
</tr>
<tr>
<td>STAT</td>
<td>2-186</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>2-188</td>
</tr>
<tr>
<td>SYSTEMQQ</td>
<td>2-188</td>
</tr>
<tr>
<td>TCLOSE</td>
<td>2-193</td>
</tr>
<tr>
<td>TIME</td>
<td>2-190</td>
</tr>
<tr>
<td>TIMEF</td>
<td>2-191</td>
</tr>
<tr>
<td>TOPEN</td>
<td>2-192</td>
</tr>
<tr>
<td>TREAD</td>
<td>2-194</td>
</tr>
<tr>
<td>TTYNAM</td>
<td>2-195</td>
</tr>
<tr>
<td>TWRITE</td>
<td>2-196</td>
</tr>
<tr>
<td>UNLINK</td>
<td>2-197</td>
</tr>
<tr>
<td>UNPACKTIMEQQ</td>
<td>2-197</td>
</tr>
<tr>
<td>portability library</td>
<td>2-1</td>
</tr>
<tr>
<td>POSIX functions</td>
<td>3-1</td>
</tr>
<tr>
<td>FFLUSH</td>
<td>3-23</td>
</tr>
<tr>
<td>FGETC</td>
<td>3-24</td>
</tr>
<tr>
<td>FPUTC</td>
<td>3-27</td>
</tr>
<tr>
<td>FSEEK</td>
<td>3-28</td>
</tr>
<tr>
<td>FTELL</td>
<td>3-30</td>
</tr>
<tr>
<td>GETC</td>
<td>3-32</td>
</tr>
<tr>
<td>GETCWD</td>
<td>3-32</td>
</tr>
<tr>
<td>IPXFARGC</td>
<td>3-2</td>
</tr>
<tr>
<td>MKDIR</td>
<td>3-49</td>
</tr>
<tr>
<td>PXFACCESS</td>
<td>3-5</td>
</tr>
<tr>
<td>PXFAINTGET</td>
<td>3-6</td>
</tr>
<tr>
<td>PXFAINTSET</td>
<td>3-7</td>
</tr>
<tr>
<td>PXFCALLSUBHANDLE</td>
<td>3-8</td>
</tr>
<tr>
<td>PXFCHDIR</td>
<td>3-9</td>
</tr>
<tr>
<td>PXFCHMOD</td>
<td>3-10</td>
</tr>
<tr>
<td>PXFCHOWN</td>
<td>3-11</td>
</tr>
<tr>
<td>PXFCLOSE</td>
<td>3-12</td>
</tr>
<tr>
<td>PXFCLOSEDIR</td>
<td>3-12</td>
</tr>
<tr>
<td>PXFCONST</td>
<td>3-13</td>
</tr>
<tr>
<td>PXFCREAT</td>
<td>3-14</td>
</tr>
<tr>
<td>PXFDUP</td>
<td>3-15</td>
</tr>
<tr>
<td>PXFDUP2</td>
<td>3-16</td>
</tr>
<tr>
<td>PXFEINTGET</td>
<td>3-16</td>
</tr>
<tr>
<td>PXFEINTSET</td>
<td>3-17</td>
</tr>
<tr>
<td>PXFESTRGET</td>
<td>3-18</td>
</tr>
<tr>
<td>PXFEXECV</td>
<td>3-19</td>
</tr>
<tr>
<td>PXFEXECVE</td>
<td>3-20</td>
</tr>
<tr>
<td>PXFEXECVP</td>
<td>3-21</td>
</tr>
<tr>
<td>PXFEXIT</td>
<td>3-22</td>
</tr>
<tr>
<td>PXFFASTEXIT</td>
<td>3-23</td>
</tr>
<tr>
<td>PXFFILENO</td>
<td>3-25</td>
</tr>
<tr>
<td>PXFFORK</td>
<td>3-26</td>
</tr>
<tr>
<td>PXFFSTAT</td>
<td>3-29</td>
</tr>
<tr>
<td>PXFGETARG</td>
<td>3-31</td>
</tr>
<tr>
<td>PXFGETGRGID</td>
<td>3-33</td>
</tr>
<tr>
<td>PXFGETGRNAM</td>
<td>3-34</td>
</tr>
<tr>
<td>PXFGETPWNAM</td>
<td>3-35</td>
</tr>
<tr>
<td>PXFGETPWUID</td>
<td>3-36</td>
</tr>
<tr>
<td>PXFGETSUBHANDLE</td>
<td>3-37</td>
</tr>
<tr>
<td>PXFINGET</td>
<td>3-38</td>
</tr>
<tr>
<td>PXFINSET</td>
<td>3-39</td>
</tr>
<tr>
<td>PXFSBLK</td>
<td>3-40</td>
</tr>
</tbody>
</table>
PXFSCHR, 3-41
PXFSCONST, 3-41
PXFSIDIR, 3-42
PXFSIFIFO, 3-43
PXFSIREG, 3-44
PXFKILL, 3-45
PXFLINK, 3-46
PXFLOWALC, 3-47
PXFLSEEK, 3-48
PXFMKFIFO, 3-50
PXFOFOPEN, 3-51
PXFOPENDIR, 3-52
PXFPipe, 3-53
PXFPUTC, 3-53
PXFREADDIR, 3-55
PXFRENAME, 3-56
PXFRHINDIR, 3-57
PXFRMDIR, 3-57
PXFSIGADDSET, 3-58
PXFSIGDELSET, 3-59
PXFSIGEMPTYSET, 3-60
PXFSIGFILLSET, 3-61
PXFSIGISMEMBER, 3-62
PXFSIGSTAT, 3-63
PXFSISTRGET, 3-64
PXFSSTRUCTCOPY, 3-65
PXFSSTRUCTCREATE, 3-66
PXFSSTRUCTFREE, 3-67
PXFSUCOMPARE, 3-67
PXFSUMASK, 3-68
PXFSUNLINK, 3-69
PXFSUTIME, 3-70
PXFSWAIT, 3-71
PXFSWAITPID, 3-71
PXFSXEXITSTATUS, 3-3
PXFSXEXITED, 3-72
PXFSXISENALED, 3-73
PXFSXISTOPPED, 3-74
PXFSWRITE, 3-74
PXFWSTOPSIG, 3-3
PXFWTERMSIG, 3-4
POSIX library, 3-1
PRECISION intrinsic function, 1-265
PRESENT intrinsic function, 1-266
procedure
intrinsic, 1-1
PRODUCT intrinsic function, 1-267
PSFFILENO POSIX subroutine, 3-25
Publications
See related publications, xxiii, xxiv
PUTC portability function, 2-139
PUTIMAGE graphic subroutine, 4-63
PUTIMAGE_W graphic subroutine, 4-65
PXFACESS POSIX subroutine, 3-5
PXFAINTGET POSIX subroutine, 3-6
PXFAINTPOSIX subroutine, 3-7
PXFCALLSUBHANDLE POSIX subroutine, 3-8
PXFCCHDIR POSIX subroutine, 3-9
PXFCCHMOD POSIX subroutine, 3-10
PXFCCHOWN POSIX subroutine, 3-11
PX FCLOSE POSIX subroutine, 3-12
PXFCLOSEDIR POSIX subroutine, 3-12
PXFCONST POSIX subroutine, 3-13
PXFCREAT POSIX subroutine, 3-14
PXFDUP POSIX subroutine, 3-15
PXFDUP2 POSIX subroutine, 3-16
PXFEINTGET POSIX subroutine, 3-16
PXFEINTSET POSIX subroutine, 3-17
PXFESTRGET POSIX subroutine, 3-18
PXFEXECV POSIX subroutine, 3-19
PXFEXECVE POSIX subroutine, 3-20
PXFEXECVP POSIX subroutine, 3-21
PXFEXIT POSIX subroutine, 3-22
PXFFASTEXIT POSIX subroutine, 3-23
PXFFORK POSIX subroutine, 3-24
PXFFSTAT POSIX subroutine, 3-29
PXFGETARG subroutine, 3-31
PXFGETGRGID POSIX subroutine, 3-33
PXFGETGRNAM POSIX subroutine, 3-34
PXFGETPWNAM POSIX subroutine, 3-35
PXGETPWUID POSIX subroutine, 3-36
PXGETSUBHANDLE POSIX subroutine, 3-37
PXFINGET POSIX subroutine, 3-38
PXFINSET POSIX subroutine, 3-39
PXFISBLK POSIX function, 3-40
PXFISCHR POSIX function, 3-41
PXFISCONST POSIX function, 3-41
PXFISDIR POSIX function, 3-42
PXFISFIFO POSIX function, 3-43
PXFISREG POSIX function, 3-44
PXFKILL POSIX subroutine, 3-45
PXFLINK POSIX subroutine, 3-46
PXFLOCALTIME POSIX subroutine, 3-47
PXFELSEEK POSIX subroutine, 3-48
PXFMKFIFO POSIX subroutine, 3-50
PXFOPEN POSIX subroutine, 3-51
PXFOPENDIR POSIX subroutine, 3-52
PXFPPIPE POSIX subroutine, 3-53
PXFPUTC POSIX subroutine, 3-53
PXFREAD POSIX subroutine, 3-54
PXFREADDIR POSIX subroutine, 3-55
PXFRENAMPOSIX subroutine, 3-56
PXFREWINDDIR POSIX subroutine, 3-57
PXFRMDIR POSIX subroutine, 3-57
PXFSIGADDSET POSIX subroutine, 3-58
PXFSIGDELSET POSIX subroutine, 3-59
PXFSIGEMPTYSET POSIX subroutine, 3-60
PXFSIGFILLSET POSIX subroutine, 3-61
PXFSIGMEMBER POSIX subroutine, 3-62
PXFSTAT POSIX subroutine, 3-63
PXFSUBGET POSIX subroutine, 3-64
PXFSUBCOPY POSIX subroutine, 3-65
PXFSUBCREATE POSIX subroutine, 3-66
PXFSUBFREE POSIX subroutine, 3-67
PXFUCOMPARE POSIX subroutine, 3-67
PXFUMASK POSIX subroutine, 3-68
PXFUNLINK POSIX subroutine, 3-69
PXFUTIME POSIX subroutine, 3-70
PXFWAIT POSIX subroutine, 3-71
PXFWAITPID POSIX subroutine, 3-71
PXFWEXITSTATUS POSIX function, 3-3
PXFWEXITED POSIX function, 3-72
PXFWFSIGNALED POSIX function, 3-73
PXFWFSTOPPED POSIX function, 3-74
PXFWRITE POSIX subroutine, 3-74
PXFWSTOPSIG POSIX function, 3-3
PXFWTERMSIG POSIX function, 3-4

Q
QRANSET portability subroutine, 2-140
QSORT portability subroutine, 2-140
QuickWin functions
ABOUTBOXQQ, 4-125
APPENDMENUQQ, 4-126
CLICKMENUQQ, 4-129
DELETEMENUQQ, 4-130
DSETGTEXTVECTOR, 4-185
FOCUSQQ, 4-131
GETACTIVEPAGE, 4-180
GETACTIVEQQ, 4-132
GETEXITQQ, 4-133
GETTEXTVECTOR, 4-181
GETHANDLECHILDQQ, 4-191
GETHANDLECLIENTQQ, 4-190
GETHANDLEFRAMEQQ, 4-190
GETHANDLEQQ, 4-181
GETHWNDQQ, 4-135
GETTEXTCURSOR, 4-180
GETUNITQQ, 4-136
GETVIDEODEVICING, 4-182
GETVISUALPAGE, 4-183
GETWINDOWCONFIG, 4-137
GETWSIZEQQ, 4-139
INQFOCUSQQ, 4-141
INSERTMENUQQ, 4-143
MESSAGEBOXQQ, 4-146
MODIFYMENUFLAGSQQ, 4-148
MODIFYMENUROUTINEQQ, 4-149
MODIFYMENUSTRINGQQ, 4-151
REGISTERFONTS, 4-183
REGISTERMOUSEEVENT, 4-152
SELECTPALETTE, 4-184
SETACTIVEPAGE, 4-184
SETACTIVEQQ, 4-154
SETEXITQQ, 4-156
SETFRAMEWINDOW, 4-185
SETMESSAGEQQ, 4-157
SETSTATUSMESSAGE, 4-186
SETTEXTCURSOR, 4-186
SETTEXTFONT, 4-187
SETTEXTROWS, 4-187
SETVIDEOMODE, 4-188
SETVIDEOMODEROWS, 4-188
SETVISUALPAGE, 4-189
SETWINDOWCONFIG, 4-159
SETWINDOWMENUQQ, 4-162
SETWSIZEQQ, 4-163
UNREGISTERFONTS, 4-189
UNREGISTERMOUSEEVENT, 4-164
UNUSEDQQ, 4-191
WAITONMOUSEEVENT, 4-166
QuickWin library, 4-1
QuickWin menu functions
NUL, 4-179
WINABOUT, 4-177
WINARRANGE, 4-174
WINCASCADE, 4-173
WINCLEARPASTE, 4-175
WINCOPY, 4-170
WINEXIT, 4-170
WINFULLSCREEN, 4-172
WININDEX, 4-176
WININPUT, 4-175
WINPASTE, 4-171
WINPRINT, 4-169
WINSAVE, 4-169
WINSELECTALL, 4-179
WINSELECTGRAPHICS, 4-178
WINSELECTTEXT, 4-178
WINSIZETOFIT, 4-171
WINSTATE, 4-172
WINSTATUS, 4-176
WIN TILE, 4-174
WINUSING, 4-177
QuickWin routines, 4-2

R
RADIX intrinsic function, 1-269
RAISEQQ portability function, 2-141
RAN portability function, 2-143
RAND intrinsic function, 2-144
RAND portability function, 2-144
RANDOM portability subroutine, 2-146
RANDOM_NUMBER intrinsic subroutine, 1-270
RANDOM_SEED intrinsic subroutine, 1-271
RANDU portability function, 2-147
RANF portability function, 2-148
RANGE intrinsic function, 1-272
RANGET portability subroutine, 2-149
RANSET portability subroutine, 2-149
real
representation of, 1-12
REAL intrinsic function, 1-273
RECTANGLE graphic function, 4-68
RECTANGLE_W graphic function, 4-69
REGISTERFONTS QuickWin function, 4-183
REGISTERMOUSEEVENT QuickWin function, 4-152
Related publications, xxiii
REMAPALLPALETTERGB graphic function, 4-70
REMAPALLPALETTERGP graphic function, 4-70
REMAPPALETTERGB graphic function, 4-72
RENAME portability function, 2-150
RENAMEFILEQQ portability function, 2-151
REPEAT intrinsic function, 1-274
RESHAPE intrinsic function, 1-275
return code, 2-44
RGBTOINTEGER color function, 4-111
RINDEX portability function, 2-152
RNUM intrinsic function, 1-277
RRSPACING intrinsic function, 1-277
RSHIFT intrinsic function, 1-278
RTC portability function, 2-153
RUNQ portability function, 2-154

S
SAVEIMAGE graphic function, 4-74
SAVEIMAGE_W graphic function, 4-75
SCALE intrinsic function, 1-279
SCAN intrinsic function, 1-280
SCANENV portability subroutine, 2-156
SCROLLTEXTWINDOW graphic subroutine, 4-78
SCWRQQ portability subroutine, 2-155
SECNDS portability function, 2-157
SEED portability subroutine, 2-157
SELECTED_INT_KIND intrinsic function, 1-281
SELECTED_REAL_KIND intrinsic function, 1-282
SELECTPALETTE QuickWin function, 4-184
SET_EXPONENT intrinsic function, 1-284
SETACTIVEPAGE QuickWin function, 4-184
SETACTIVEQQ QuickWin function, 4-154
SETBKCOLOR graphic function, 4-79
SETBKCOLORRGB color function, 4-107
SETCLIPRGN graphic subroutine, 4-80
SETCOLOR graphic function, 4-81
SETCOLORRGB color function, 4-105
SETCONTROLFPQQ portability subroutine, 2-159
SETDAT portability subroutine, 2-161
SETENVQQ portability function, 2-162
SETERRORMODEQQ portability subroutine, 2-164
SETEXITQQ QuickWin function, 4-156
SETFILEACCESSQQ portability function, 2-166
SETFILETIMEQQ portability function, 2-167
SETFILLMASK graphic subroutine, 4-82
SETFONT font manipulation function, 4-117
SETFRAMEWINDOW QuickWin subroutine, 4-185
SETGTEXTROTATION font manipulation subroutine, 4-120
SETLINESTYLE graphic subroutine, 4-83
SETMESSAGEQQ QuickWin subroutine, 4-157
SETPIXEL graphic function, 4-84
SETPIXEL_W graphic function, 4-85
SETPIXELRGB color function, 4-108
SETPIXELRGB pixel function, 4-108
SETPIXELRGB_W color function, 4-110
SETPIXELRGB_W pixel function, 4-110
SETPIXELS graphic subroutine, 4-86
SETPIXELSRGB color subroutine, 4-102
SETPIXELSRGB pixel subroutine, 4-102
SETSTATUSMESSAGE QuickWin subroutine, 4-186
SETTEXTCOLOR graphic function, 4-87
SETTEXTCOLORRGB font manipulation function, 4-123
SETTEXTCURSOR QuickWin function, 4-186
SETTEXTFONT QuickWin subroutine, 4-187
SETTEXTPosition graphic subroutine, 4-88
SETTEXTROWS QuickWin function, 4-187
SETTEXTWINDOW graphic subroutine, 4-89
SETTIM portability subroutine, 2-169
SETVIDEOMODE QuickWin function, 4-188
SETVIDEOMODEROWS function, 4-188
SETVIEWORG graphic subroutine, 4-90
Index

SETVIEWPORT graphic subroutine, 4-91
SETVISUALPAGE QuickWin function, 4-189
SETWINDOW graphic function, 4-92
SETWINDOWCONFIG QuickWin function, 4-159
SETWINDOWMENUQQ QuickWin function, 4-162
SETWRITEMODEgraphicfunction, 4-93

subroutines
CLEARSCREEN graphic subroutine, 4-16
CLOCKX portability subroutine, 2-20
DATE portability subroutine, 2-26
DATE4 portability subroutine, 2-27
DRANSET portability subroutine, 2-39
DSETGTEXTVECTOR QuickWin subroutine, 4-185
FDATE portability subroutine, 2-45
FGETC POSIX subroutine, 3-24
FGETCPOSIX subroutine, 3-24
FLLUSH POSIX subroutine, 2-26
FLLUSHposix subroutine, 2-27
FPUTC POSIX subroutine, 3-27
FSEEK portability subroutine, 2-52
FSEEK POSIX subroutine, 3-28
FTELL POSIX subroutine, 3-30
GERNO portability subroutine, 2-61
GETARG portability subroutine, 2-62
GETC POSIX subroutine, 3-32
GETCURRENTPOSITION graphic subroutine, 4-29
GETCURRENTPOSITION_W graphic subroutine, 4-30
GETCWD POSIX subroutine, 3-32
GETDAT portability subroutine, 2-70
GETENV portability subroutine, 2-76
GETFILLMASK graphic subroutine, 4-31
GETGTEXTVECTOR QuickWin subroutine, 4-181
GETIMAGE graphic subroutine, 4-32
GETIMAGE_W graphic subroutine, 4-33
GETINFO portability subroutine, 2-87
GETLOG portability subroutine, 2-87
GETLOGCOORD graphic subroutine, 4-35
GETPIXELS graphic subroutine, 4-38
GETPIXELSRGB pixel and color subroutine, 4-104
GETTEXTPOSITION graphic subroutine, 4-40
GETTEXTWINDOW graphic subroutine, 4-40
GETTIME portability subroutine, 2-93
GETTIMEOFDAY portability subroutine, 2-94

statement functions
naming conflicts, 1-2

statements
INTRINSIC, 1-7

subroutine
elemental intrinsic, 1-3
intrinsic, 1-3

SHAPEx intrinsic function, 1-285
SHIFTL portability function, 2-170
SHIFTR portability function, 2-171
SHORT portability function, 2-172
SIGN intrinsic function, 1-286
SIGNAL portability function, 2-173
SIGNAQ portability function, 2-176
SIN intrinsic function, 1-286
SIND intrinsic function, 1-287
SINH intrinsic function, 1-288
SIZE intrinsic function, 1-289
SLEEP portability subroutine, 2-179
SLEEPQ portability subroutine, 2-179
SORTQ portability subroutine, 2-180
SPACING intrinsic function, 1-290
specific intrinsic function, 1-4, 1-15
SPLITPATH portability subroutine, 2-182
SPREAD intrinsic function, 1-291
SQR intrinsic function, 1-292
SRAND portability subroutine, 2-184
SSWRFQ portability subroutine, 2-185
STAT portability function, 2-186
statement functions
naming conflicts, 1-2

statements
INTRINSIC, 1-7

subroutine
elemental intrinsic, 1-3
intrinsic, 1-3
GETVIDEOCONFIG QuickWin subroutine, 4-182
GETVIEWCOORD graphic subroutine, 4-41
GETVIEWCOORD_W graphic subroutine, 4-42
GETWINDOWCOORD graphic subroutine, 4-43
GMTIME portability subroutine, 2-96
HOSTNM portability subroutine, 2-98
IDATE portability subroutine, 2-101
IDATE4 portability subroutine, 2-102
IRANGET portability subroutine, 2-116
IRANSET portability subroutine, 2-117
ITIME portability subroutine, 2-118
JDATE portability subroutine, 2-119
JDATE4 portability subroutine, 2-120
LTIME portability subroutine, 2-126
MKDIR POSIX subroutine, 3-49
MOVETO graphic subroutine, 4-54
MOVETO_W graphic subroutine, 4-55
NLSGetLocale inquiry subroutine, 2-207
NUL QuickWin menu subroutine, 4-179
OUTGTEXT font manipulation subroutine, 4-116
OUTTTEXT graphic subroutine, 4-56
PERROr portability subroutine, 2-137
PSFLINK POSIX subroutine, 3-46
PUTIMAGE graphic subroutine, 4-63
PUTIMAGE_W graphic subroutine, 4-65
PXFACCESS POSIX subroutine, 3-5
PXFAINTGET POSIX subroutine, 3-6
PXFAINTSET POSIX subroutine, 3-7
PXFCALLSUBHANDLE POSIX subroutine, 3-8
PXFCHDIR POSIX subroutine, 3-9
PXFCHMOD POSIX subroutine, 3-10
PXFCHOWN POSIX subroutine, 3-11
PXFCLOSE POSIX subroutine, 3-12
PXFCLOSEDIR POSIX subroutine, 3-12
PXFCONST POSIX subroutine, 3-13
PXFCREATE POSIX subroutine, 3-14
PXFDUP POSIX subroutine, 3-15
PXFDUP2 POSIX subroutine, 3-16
PXFEINTGET POSIX subroutine, 3-16
PXFEINTSET POSIX subroutine, 3-17
PXFEEXECV POSIX subroutine, 3-19
PXFEEXECVE POSIX subroutine, 3-20
PXFEEXECVP POSIX subroutine, 3-21
PXFEEXIT POSIX subroutine, 3-22
PXFFASTEXIT POSIX subroutine, 3-23
PXFFILENO POSIX subroutine, 3-25
PXFFORK POSIX subroutine, 3-26
PXFFSTAT POSIX subroutine, 3-29
PXFGETARG POSIX subroutine, 3-31
PXFGETGRGID POSIX subroutine, 3-33
PXFGETGRNAM POSIX subroutine, 3-34
PXFGETPWDNAM POSIX subroutine, 3-35
PXFGETPWDUID POSIX subroutine, 3-36
PXFGETSUBHANDLE POSIX subroutine, 3-37
PXFINTGET POSIX subroutine, 3-38
PXFINTSET POSIX subroutine, 3-39
PXFKILL POSIX subroutine, 3-45
PXFLOCALTIME POSIX subroutine, 3-47
PXFSEEK POSIX subroutine, 3-48
PXFMKIFO POSIX subroutine, 3-50
PXFOPEN POSIX subroutine, 3-51
PXFOPENDIR POSIX subroutine, 3-52
PXFPPIPE POSIX subroutine, 3-53
PXFPUTC POSIX subroutine, 3-53
PXFRREAD POSIX subroutine, 3-54
PXFRADDIR POSIX subroutine, 3-55
PXFRNAME POSIX subroutine, 3-56
PXFRWINDDIR POSIX subroutine, 3-57
PXFRMDIR POSIX subroutine, 3-57
PXFSIGADDSET POSIX subroutine, 3-58
PXFSIGDELSSET POSIX subroutine, 3-59
PXFSIGEMPSETYPOSIX subroutine, 3-60
PXFSIGFILLSET POSIX subroutine, 3-61
PXFSIGISMEMBER POSIX subroutine, 3-62
PXFSTAT POSIX subroutine, 3-63
PXFRSTRGET POSIX subroutine, 3-64
PXFSTRUCTCREATE POSIX subroutine, 3-66
PXFSTRUCTFREE POSIX subroutine, 3-67
PXFSSTRUCTOPY POSIX subroutine, 3-65
PXFFUCOMPARE POSIX subroutine, 3-67
PXFFUMASK POSIX subroutine, 3-68
PXFFUNLINK POSIX subroutine, 3-69
PXFFUTIME POSIX subroutine, 3-70
PXFFWAIT POSIX subroutine, 3-71
PXFWAITPID POSIX subroutine, 3-71
PXFWRITE, 3-74
QRANSET portability subroutine, 2-140
QSORT portability subroutine, 2-140
RANDOM portability subroutine, 2-146
RANGET portability subroutine, 2-149
RANSET portability subroutine, 2-149
SCANENV portability subroutine, 2-156
SCROLLTEXTWINDOW graphic subroutine, 4-78
SEED portability subroutine, 2-157
SETCLIPRGN graphic subroutine, 4-80
SETDAT portability subroutine, 2-161
SETTEXTFONT QuickWin subroutine, 4-187
SETFILLMASK graphic subroutine, 4-82
SETFRAMEWINDOW QuickWin subroutine, 4-185
SETGTEXTRotation font manipulation subroutine, 4-120
SETLINESTYLE graphic subroutine, 4-83
SETMESSAGEQQ QuickWin subroutine, 4-157
SETPIXELS graphic subroutine, 4-86
SETPIXELSRGB pixel and color subroutine, 4-102
SETSTATUSMESSAGE QuickWin subroutine, 4-186
SETTEXTEXTPOSITION graphic subroutine, 4-88
SETTEXTEXTWINDOW graphic subroutine, 4-89
SETTIM portability subroutine, 2-169
SETVIEWORG graphic subroutine, 4-90
SETVIEWPORT graphic subroutine, 4-91
SLEEP portability subroutine, 2-179
SRAND portability subroutine, 2-184
TIME portability subroutine, 2-190
UNREGISTERFONTS QuickWin subroutine, 4-189
UNSEDFSQ QuickWin window handle subroutine, 4-191
WINABOUT QuickWin menu subroutine, 4-177
WINARRANGE QuickWin menu subroutine, 4-174
WINCASCADE QuickWin menu subroutine, 4-173
WINCLEARPASTE QuickWin menu subroutine, 4-175
WINCOPY QuickWin menu subroutine, 4-170
WINEXIT QuickWin menu subroutine, 4-170
WINFULLSCREEN QuickWin menu subroutine, 4-172
WININDEX QuickWin menu subroutine, 4-176
WININPUT QuickWin menu subroutine, 4-175
WNPASTE QuickWin menu subroutine, 4-171
WINPRINT QuickWin menu subroutine, 4-169
WINSAVE menu subroutine, 4-169
WINSELECTALL QuickWin menu subroutine, 4-179
WINSELECTGRAPHICS QuickWin menu subroutine, 4-178
WINSELECTTEXT QuickWin menu subroutine, 4-178
WINSIZETOFIT QuickWin menu subroutine, 4-171
WINSTATE QuickWin menu subroutine, 4-172
WINSTATUS QuickWin menu subroutine, 4-176
WINTILE QuickWin menu subroutine, 4-174
WINUSING QuickWin menu subroutine, 4-177
SUM intrinsic function, 1-293
syntax
  intrinsic procedure. Chapter 11, 1-1
SYSTEM portability function, 2-188
SYSTEM_CLOCK intrinsic subroutine, 1-295
SYSTEMQQ portability FUNCTION, 2-188

T
TAN intrinsic function, 1-296
TAND intrinsic function, 1-297
TANH intrinsic function, 1-298
Target architecture, xxiv
TCLOSE portability function, 2-193
time for program execution, 1-248
TIME portability subroutine, 2-190
TIMEF portability function, 2-191
TINY intrinsic function, 1-299
TOPEN portability function, 2-192
TRANSFER intrinsic function, 1-300
transformational function, 1-6
TRANSPOSE intrinsic function, 1-302
TREAD portability function, 2-194
TRIM intrinsic function, 1-303
TTYNAM portability function, 2-195
TWRITE portability function, 2-196

U
UBOUND intrinsic function, 1-304
UNLINK portability function, 2-197
UNPACK intrinsic function, 1-306
UNPACKTIMEQQ portability subroutine, 2-197
UNREGISTERFONTS QuickWin subroutine, 4-189
UNREGISTERMOUSEEVENT QuickWin function, 4-164
UNUSEDQQ QuickWin window handle subroutine, 4-191
USE IFLPORT statement, 2-2

V
VERIFY intrinsic function, 1-308

W
WAITONMOUSEEVENT QuickWin function, 4-166
WINABOUT QuickWin menu subroutine, 4-177
WINARRANGE QuickWin menu subroutine, 4-174
WINCASCADE QuickWin menu subroutine, 4-173
WINCLEARPASTE QuickWin menu subroutine, 4-175
WINCOPY QuickWin menu subroutine, 4-170
WINEXIT Quickwin menu subroutine, 4-170
WINFULLSCREEN QuickWin menu subroutine, 4-172
WININDEX QuickWin menu subroutine, 4-176
WININPUT QuickWin menu subroutine, 4-175
WINPASTE QuickWin menu subroutine, 4-169
WINSAVE QuickWin menu subroutine, 4-169
WINSELECTALL QuickWin menu subroutine, 4-179
WINSELECTGRAPHICS QuickWin menu subroutine, 4-178
WINSELECTTEXT QuickWin menu subroutine, 4-178
WINSIZETOFIT QuickWin menu subroutine, 4-171
WINSTATE QuickWin menu subroutine, 4-172
WINSTATUS QuickWin menu subroutine, 4-176
WINTILE QuickWin menu subroutine, 4-174
WINUSING QuickWin menu subroutine, 4-177
WRAPON graphic function, 4-95

X
XOR intrinsic function, 1-310